Algorithmes et architectures efficaces pour vision embarquée

Eva Dokladalova

Université Paris-Est

Mémoire en vue d'obtention de

Habilitation à Diriger des Recherches

Spécialité : Mathématiques et STIC - traitement du signal et des images

Algorithmes et architectures efficaces pour vision embarquée

dossier présenté par

Eva DOKLADALOVA

préparé pour une soutenance publique le 21 juin 2019

devant le jury composé de:

Jury

Mme. Fan YANG,	Professeur, Université de Bourgogne	Rapporteure
M. Jean-François NEZAN,	Professeur, INSA Rennes	Rapporteur
M. Antoine DUPRET,	HDR, chercheur CEA-LIST	Rapporteur
Mme. Beatriz MARCOTEGUI,	Professeur, Mines-ParisTech	Examinatrice
M. Carlos VALDERRAMA,	Professeur, Université de Mons	Examinateur
M. Antoine MANZANERA,	Professeur, ENSTA	Examinateur
M. Laurent GEORGE,	Professeur, ESIEE Paris	Examinateur

ESIEE Paris, Université Paris-Est
UMR CNRS 8049, Noisy le Grand, France

Remerciements

Avant tout je souhaiterais remercier les membres du jury pour l'honneur qu'ils m'accordent par leur participation. Merci en particulier aux rapporteurs pour leurs commentaires et leurs critiques.

Je souhaiterais également addresser mes remerciments à tous les chercheurs, ensigenants-chercheurs, ingénieurs, doctorands et stagiaires avec lesquels j'ai eu la chance de travailler à l'Ecole des Mines, au CEA-LIST, à l'ESIEE Paris ou LIGM et qui m'ont tous apporté de l'inspiration et de la motivation.

Merci à tous les membres du Département informatique de l'ESIEE Paris pour leur sympathie et l'ambiance agréable.

Mes derniers remerciements, mais pas les moindres vont à ma famille, Petr, Hugo et Maxim qui n'ont jamais douté de moi.

A Petr, Hugo et Maxim.

Table des matières

Chapitre 1

Introduction

Aujourd'hui, j'ai 14 ans d'expérience en recherche, en encadrement et 12 ans en enseignement. Depuis certain temps, j'effectue une rétrospective de mes travaux de recherche non seulement pour mettre en évidence leur cohérence, mais aussi pour faire ressortir des nouveaux objectifs scientifiquement pertinents par rapport à l'évolution du contexte scientifique actuel.

Dans ce contexte, je vois la préparation d'un diplôme de l'habilitation à diriger des recherches non seulement comme un objectif majeur d'une carrière scientifique et mais aussi comme un excellent moyen permettant de réaliser ce bilan.

1.1 Parcours personnel

Étant armée d'un DEA [1] en automatique de l'Institut National Polytechnique de Grenoble, obtenu en 2000, j'ai décidé de poursuivre ma formation par recherche en préparant un doctorat dans le domaine des architectures matérielles pour le traitement d'image temps-réel ; plus précisément pour les méthodes basées sur des équations aux dérivées partielles. La confrontation entre un problème mathématique complexe et les limites d'organisation des supports de calculs me paraissait un défi à relever.

Assez rapidement, j'ai été confrontée non seulement au problème de puissance de calcul, mais aussi aux problèmes algorithmiques, problèmes du domaine de parallélisme, de structures de données, mais également de développement des applications utilisant des modèles étudiés, le plus souvent en lien avec le monde des systèmes de vision embarquée et toujours avec l'objectif d'améliorer des multiples caractéristiques d'un tel système.

Puis, pendant plusieurs années passées au Laboratoire de Calculateurs Embarqués et Image (LCEI) au CEA-LIST [2] à Saclay, j'ai pu acquérir une expérience de recherche partenariale dans le domaine des technologies de vision dédiée à la sécurité automobile.

Depuis l'arrivée à l'ESIEE Paris, mes activités se sont enrichies de la supervision de doctorants et des collaborations académiques. Dans le passé, j'ai co-supervisé 5 doctorants et, actuellement, je co-encadre 3 doctorants. J'ai également continué à enrichir ma "polycompétence", mais l'accélération du rythme actuel de recherche m'oblige à focaliser de plus en plus sur le côté algorithmique et applicatif.

Je suis passionnée par le développement des nouvelles idées à travers des coopérations scientifiques extérieures où j'apprécie les collaborations pluridisciplinaires permettant de découvrir des problèmes et des solutions inattendus et innovants.

1. DEA - Diplôme d'études approfondies
2. Commissariat à l'énergie atomique et aux énergies alternatives

En fin, j'affectionne beaucoup la transmission du savoir par l'encadrement de jeunes chercheurs en créant une ambition d'équipe et de synergie des thèmes.

Actuellement, je suis professeur associé et mes activités d'enseignement passées et actuelles couvrent quatre domaines principaux : traitement d'images, systèmes de vision intégrés, architecture des ordinateurs et programmation, implémentations parallèles. Depuis plusieurs années, je participe à l'enseignement dans la filière Internationale, appelée Master of Computer Science, enseignée entièrement en anglais. Je participe également au master de recherche (M2) Science de l'image où l'ESIEE Paris est co-habilité avec l'Université Paris-Est.

Durant les 7 dernières années, j'ai exercé la co-responsabilité de la filière Informatique. Actuellement, je m'investis dans l'organisation de la mobilité internationale de nos élèves, en assurant le rôle du Correspondant des projets internationaux.

1.2 Organisation du manuscrit

Pour présenter mes contributions scientifiques et académiques de manière plus approfondie, ce document contient des parties suivantes :

- Présentation du Curriculum Vitae (Chapitre 2) et des activités scientifiques (Chapitre 3), pédagogiques (Chapitre 5) et institutionnelles (Chapitre 4), accompagnées de la liste exhaustive des communications scientifiques (Chapitre 6) ;

- Contributions algorithmiques (Partie II) : Morphologie mathématique continue (Chapitre 7) - calcul massivement parallèle et asynchrone de la fonction distance, avec une précision sous-pixellique (2D et 3D) ; Morphologie mathématique discrète (Chapitre 8) - algorithme de calcul des opérateurs concaténés en temps constant, avec l'utilisation de mémoire et la latence de calcul optimale pour des filtres de grandes tailles et angles arbitraires ;

- Implantations efficaces (Partie III) : à chaque contribution algorithmique est liée au moins une innovation technologique, c'est-à-dire soit une nouvelle réalisation matérielle efficace ou soit une réalisation parallèle sur des processeurs multi-cœurs ou GPU [3]. Chacune de ces propositions atteignait des performances dépassant l'état de l'art à la date de la publication : calcul globalement asynchrone de l'algorithme de ligne de partage des eaux (LPE) permettant de simplifier radicalement la gestion d'accès aux données tout en améliorant le temps d'exécution (Chapitre 9) ; proposition d'un coprocesseur morphologique à latence de calcul minimale (Chapitre 10) ; réalisation matérielle dédiée pour le calcul des opérateurs de traitement d'image formulés à l'aide de l'arbre de composantes connexes (min-max tree). Une telle réalisation a été considérée comme impossible auparavant (Chapitre 11).

- Transfert technologique et applications (Partie IV) : certaines de ces contributions ont permis des innovations industrielles : conception et validation d'un processeur embarqué reconfigurable pour la sécurité automobile (systèmes ADAS) (FIAT, FICOSA, ST-Microelectronics) (Chapitre 12) ; architecture matérielle auto-adaptable pour des systèmes de vision multi-capteurs (SAFRAN-Defense) (Section 12.2) ; détection, reconnaissance et classification des particules pour dosimètre intelligent avec le capteur TimePix (IPEA [4], Prague) (Section 13.2) ; détection et reconnaissance des cibles pour UAV [5] (Section 13.1).

En fin, le manuscrit se termine par la présentation de mon projet de recherche et de l'enseignement (Partie V).

3. GPU - Graphics Processing Unit
4. Institut de Physique Expérimentale et Appliquée
5. UAV-Unmanned Aerial Vehicles

Première partie

Curriculum Vitæ, résumé des activités

Chapter 2

Curriculum vitae

Eva Dokládalová

Associate professor

ESIEE Paris
Computer science department
2 boulevard Blaise Pascal
Cité Descartes, BP 99
93162 Noisy-le-Grand Cedex
Tel. : + 33 (0) 1 45 92 60 39
E-mail : eva.dokladalova@esiee.fr

PERSONAL DATA

Birth name: Dejnožková
Date and Place of Birth: **November 2nd 1975**, **Most, Czech Republic**
Nationality: **Czech Republic**
Languages:
- **Czech** - mother's tongue
- **French** - fluent (bilingual)
- **English** - complete professionnal skills
- **Spanish** - basics

Marital status: **Married, two children**

EDUCATION

2000 - 2004 **PhD** from **Ecole Nationale Supérieure des Mines de Paris** (MINES ParisTech), France, http://www.mines-paristech.fr/
Title: *Dedicated architectures for image processing methods based on partial differential equations*,
Hosting laboratory: Center of Mathematical Morphology,
Advisor: Jean-Claude Klein
With congratulations of jury.
Award of University of West Bohemia Rector for outstanding PhD

1999 - 2000 **Diplôme d'études approfondies (DEA)**, from **Institut National Polytechnique de Grenoble, France** (Grenoble INP), http://www.grenoble-inp.fr/
option Control systems, Stand-alone postgraduate degree aimed to prepare doctoral studies,
Graduated with honours, rank 1/25

1994 - 1999 **Engineer in Applicated Electronics**, from **Faculty of Electrotechnical Engineering, University of West Bohemia, Plzen**, Czech Republic
Graduated with honours, rank 1/86

APPOINTMENTS

since 2006 on **Lecturer and researcher**, Computer science department, ESIEE Paris
- from 2018 **Correspondent for international projects**
- 02/18-07/18 - **Research period**, Mines-ParisTech, New vision methods for UAVs
- from 2011 - **Associate professor**, Computer science department
- 2010 - 2018 **Head of computer science track** for the 5^{th} year of engineering study
- from 2007 **member of Laboratoire d'Informatique Gaspard-Monge UMR CNRS 8049**

[10pt] 2005 - 2006 **Full time researcher**, Embedded computing and image laboratory of CEA Saclay, France
EU MEDEA+ funding : *Conception and design of SoC for automotive applications*

2004 - 2005 **Post doctoral researcher**, Embedded computing and image laboratory of CEA Saclay, France
Internal research funding : *MPEG-4 AVC in embedded systems, dimensionning and design*

2004 (Jan - March) **Research assistant**, Center of Mathematical Morphology, Ecole Nationale Supérieure des Mines de Paris, France

REFERENCES

Laurent GEORGE, HDR	Professor at ESIEE Paris Head of Computer Science Departement ESIEE Paris Laboratoire d'informatique Gaspard Monge, UMR 8049 2 boulevard Blaise Pascal, Cité Descartes, BP 99, 93162 Noisy-le-Grand laurent.george@esiee.fr
Doc. Dr. Ing. Vjaceslav GEORGIEV	Head of Applied Electronics Departement Vice-Dean for International Relations West Bohemia University, Pilsen, Czech republic Univerzitni 8, 306 14 Pilsen, Czech republic georg@kae.zcu.cz
Jean-François BERCHER, HDR	Professor at ESIEE Paris Dean of ESIEE Paris Laboratoire d'informatique Gaspard Monge, UMR 8049 2 boulevard Blaise Pascal, Cité Descartes, BP 99, 93162 Noisy-le-Grand jf.bercher@esiee.fr

Chapter 3

Research Activities

3.1 Algorithms and architectures for computer vision

Since 2000 - the beginning of my doctoral thesis - I develop a research theme related to portable or mobile vision systems where application demand is extremely strong and involves new scientific challenges: real-time processing of video streams in resolution exceeding HD [1], increase in the number of sensors within the same system and support various functions. To meet these challenges, I contribute to three fundamental axis of the field of image processing: reduced complexity algorithms for mathematical morphology, parallel and dedicated realizations for embedded vision systems and object analysis and recognition in various embedded vision systems applications.

Here I complete with lists of significant contributions.

3.1.1 Algorithms with reduced complexity for mathematical morphology

2000-2004 *Massive Marching*,
first massively parallel and globally asynchronous algorithm for the computation of the weighted distance function, defined on the basis of the partial differential equations, allowing to obtain a sub-pixel precision [27], [29], [54].

2007-2008 *Spatially variant morphological operators*,
approximation of concatenated operators, computed in constant time, with linear computation complexity [32], [13].

2007-2010 *Connected component tree*,
proposal of a new data structure and a parallel algorithm for the computation of the tree of connected components (min-max tree), allowing to maximize the degree of computational parallelism [15].

2010-2015 *Peak elimination*,
new algorithm with linear complexity for parallel computation of the openings / closings and morphological granulometries in constant time compared to the size and the number of operators of the filter [19], [7]

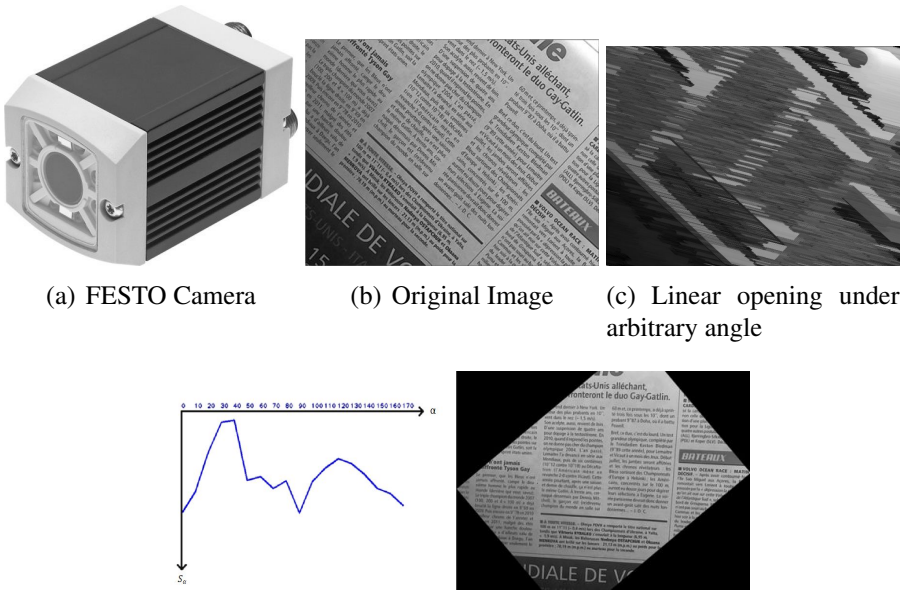
2008-2014 *Large morphological concatenated operators and arbitrary angle*
an original algorithm allowing to obtain long concatenations of the morphological atomic operators in constant time, independent of the angle or size [6], [2], [23]

1. HD - High Definition

— PhD Theses, involved in this axis: P. Matas, J. Bartovský

— Research internships involved in this area: D. Schneider, P. Karas, P. Siška, C. Clienti

— Demonstration example: Orientation detection and rotation of large-format text in real time, implanted and embedded in a FESTO industrial camera.



(a) FESTO Camera (b) Original Image (c) Linear opening under arbitrary angle

(d) Distribution of angles (e) Rotated image

Figure 3.1 – Orientation detection and rotation of large-format text in real time

3.1.2 Intelligent hardware architectures for embedded or portable vision

Work on adaptable architectures was initiated as part of Nicolas Ngan's PhD (industrial partnership with SAFRAN Defense) and since 2014 they have been pursued in Ali Isavudeen's thesis. Their work is in the context of military equipment, multi-sensor and heterogeneous. Note that these applications involve strict constraints in terms of execution time, latency, computation and congestion.

This work is inspired from then the top level research experience in the field of embedded architectures that I carried out in the CEA Embedded Calculator and Image Laboratory.

2000-2004 *Multi-processor architecture for image processing based on partial differential equations*,
first hardware architecture implementing at the same time low and high level operators of image processing based on derivative equations partial: morphological filters, linear and nonlinear diffusion filters, segmentation by geodesic distance [28], [5], [53]

2004-2005 *Design of a heterogeneous hardware architecture for the H.264 encoder (MPEG4-AVC)*
Study and estimation of algorithmic completeness, study of the hardware implementation cost, dedicated implementation of the control of a heterogeneous architecture [33].

2005-2006 *Reconfigurable System-on-Chip for Driving Assistance Vision System*, [34], [57], [56]

2007-2011 *Dynamically adaptive network on chip (NoC) for multi-sensor system* [40], [41], [43]

2007-2010 *Hardware architecture dedicated to the parallel construction of the associated component tree*, the first hardware architecture allowing a partial connected component (per line) tree construction in near linear time [42], [39].

2008-2015 *Morphological coprocessor, programmable hardware accelerator*, specialized in processing with large structuring elements. This processing unit is programmable from C or Python language, and provides unparalleled performance [18], [1].

2012-2015 *P2P: Programmable Pipeline Image Processor*, proposal of a new systolic, programmable, low-power processor for pixel flow processing, in collaboration with University of Mons, Belgium [9].

2010-2017 *Self-adaptable and conscious architecture for multi-sensor vision system*, [36], [46], [14].

Summary of obtained results

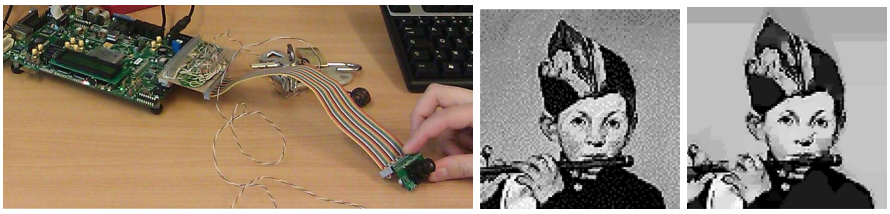
— PhD Theses, involved in this axis : P. Matas, N. Ngan, J. Bartovsky, P. Possa, A. Isavudeen

— Research internships, involved in this axis : J. Krawcyk, A. Kachkach, C. Clienti

— Patents : 2

— Demonstration example: Real-time detection and recognition of license plates on a Xilinx FPGA, Virtex 6, demonstration of calculation of concatenated operators in constant time.

■



(a) Prototyping board (b) Original image (c) Image filtered by alternating sequential filter

Figure 3.2 – Demonstration of computing concatenated operators in constant time.

3.1.3 Industrial applications and research collaboration

2002-2004 *Application of image processing methods based on partial differential equations*, face tracking for real-time hypo-vigilance detection, analysis of medical images (retinal vessels), qualification of materials [31], [12],[38]

2005-2006 *Motion Detection, Obstacle / Pedestrian Detection, ADAS System Functionality* [2], [34],[11],[56]

2. ADAS - Automotive Driving Assistance System

2010-2011 *Smart Dosimeter*, Detection and Particle Classification of the Medipix / Timepix Detector [20] This hardware implementation was also used to implement the application of morphological classification of charged particles, acquired by the TIMEPIX detector. We were able to process up to 750 frames/s in standard sensor resolution (256x256pixels), with a depth of 14bits per pixel.

2015-2018 *On-the-fly scene comprehension, in the data flow*, it relates to the recognition of the scene by the low-level morphological tools that could be entirely focused on the proposed morphological accelerator. (In progress) [9]

Summary of obtained results

— PhD theses, involved in this area: J. Bartovsky, P. Possa

— Research interns involved in this area: D. Schneider, G. Infantino, A. Baquedano, S. Sivanantham

— Demonstration example: Target detection and recognition for autonomous flying vehicles (UAV [a])

■

a. UAV - Unmanned Aerial Vehicles

(a) Example of flight plan (b) Target detected at 13m (c) Target detected at 4m

Figure 3.3 – Target detection and tracking for automatic landing of drones.

3.2 Undergoing research

3.2.1 Vision Methods for Navigation of Unmanned Aerial Vehicles

The technological bottleneck of large industrial deployment of automated UAV is due to their limited understanding of the environment. To be reliable, an UAV needs to be aware of its environment. It means to have the capability to understand the observed scene and to manage its operation with respect to the constraints coming from this scene analysis. The major problem of the state of the art scene understanding techniques is that they require very specific computing approaches, non-negligeable computational capabilities thus limiting the applicative domain.

In this research work, we aim at developing a new framework for embedded machine vision for complex scene understanding able to provide an aid to automated driving decision chain (trajectory definition), either where GPS is unavailable or where the GPS-driven coarse navigation is lacking accuracy.

Summary

— PhD theses, involved in this area: Elias Barbudo (ESIEE-UPE-CONACYT), Eric Bazan (Mines-ParisTech)

— Demonstration example: Demostration - Non supervised perceptual model for target recognition in UAVs (click on the link)

— First results: [45, 10]

— In preparation: [60]

■

3.2.2 Reduced model representation

At present, the deep learning starts to outperform the state-of-the-art accuracy on many AI tasks, it comes at the cost of high computational complexity. One of the reasons is that the performance of these networks can be increased by increasing the size of the networks. Accordingly, designing efficient and reduced models for AI[3] computing algorithms is an important step towards enabling the wide deployment of efficient neural networks (such as DNN) in AI systems, especially in embedded systems and connected objects.

In this work, we explore the utilisation of hand-engineered descriptors with rotation and translation invariant mapping. We focus on the minimization of the accuracy loss while providing faster learning (10minutes vs 3 hrs) using smaller databases (50k images vs 10M images), rotation and translation invariant representation of smaller model (40% less memory usage)

Summary

— PhD these, involved in this area: Rosemberg Oswaldo Rodrigez Salas (ESIEE-UPE)

— Master these, involved in this area: Steve Sivanhathan

— Demonstration example: under blind review for BMVC, will be published early

— First results: [62]

— In preparation: [61, 58]

■

3.2.3 Adaptable Embedded Multi-Processing for Time Critical Vision Applications

The objective of this research theme is to develop a user-friendly mapping methodology for CGPA[4]. We focus on Embedded vision systems facing the time-critical execution constraints, mainly in the terms of latency. We consider adaptable or programmable computing support with multi-processing capabilities.

The methodology should be able to efficiently map and schedule any application suited for this hardware support. It needs to provide a performance estimation (latency estimation) of the implementation and the configuration context.

The mapping methodology targets online (runtime) generation of the execution context. For this purpose, we require a precise but efficient model of the application and the hardware. At the present work status, we propose a methodology based on three graph-based models (DAG[5]) and a new mapping algorithm.

3. Artificial Intelligence
4. Coarse-Grain Programmable Architecture
5. Directed Acyclic Graph

Our approach represents the compete for framework since we consider the configuration of the system and the memory interface management.

Also, we aim to develop a demonstrator, recreating real industrial conditions. This demonstrator will be used in the DiXite project.

Summary

— PhD these, involved in this area: Elias Barbudo

— DiXite project : Machine vision for construction sites

— First results: [17, 16]

— In preparation: [59]

3.3 Research Evaluation

To evaluate the quality of research, it is usual on the one hand to quantify the results obtained in terms of the number and type of scientific papers and to calculate the quality indicators, based on the number of citations of the latter. The evaluation of my research can be quantified as follows:

Summary

— participation on supervision of 8 PhD students : 5 defended, 3 in progress
  — out of which 2 industrial partnership thesis (SAGEM DEFENSE)
  — out of which 2 supervision collaborations with Mines-ParisTech
  — out of which 1 with the membership in the accompanying committee with University of Mons

— 50 research papers,
  — 9 international journal papers
  — more than 30 articles in international refereed conferences with review
  — 5 guest lectures
  — 2 patents

— h-index = 9, according to the Google Scholar

— 287 citations

To analyse these indicators, it is necessary to consider the scientific context that is specific to the field and the work environment:

— In the field of architectures dedicated to embedded image processing, the scientific community is restricted (number of readers limited → limited number of citations)

— Research in the hardware architectures field requires a development time more than 10x greater than the software programming. Comparative studies are complex considering the variety of technologies used by the authors.

— The results presented are located in a context of strong industrial collaboration that both inspires research but is often limiting for the publication effort.

— The research/teaching activity represents 40/60% at my institution, the 40% includes PhD student supervision, master thesis supervision, grant submissions,...

3.4 Supervision

(Co-)supervised defended thesis

2011 Nicolas Ngan, "Study and design of dynamically adaptable on-chip network for embedded vision". Thesis CIFRE with SAGEM Defense, January 2008 - December 2011. Thesis supervisor: Mohamed Akil, co-director: Eva Dokladalova

2012 Jan Bartovsky, "Hardware Architectures for Morphological Filters with Large Structuring Elements". Thesis in co-supervision with the University of West Bohemia in Pilsen, Czech Republic, October 2009 - November 2012. PhD supervisors: Mohamed Akil and Vjaceslav Georgiev, co-director: Eva Dokladalova; **Eiffel Excellence Scholarship**

2013 Paulo Ricardo da Cunha Possa, Reconfigurable low-latency architecture for real-time image and video processing, **participation in the thesis support committee and the defense board (June 2013)**. Polytechnic Faculty of Mons University, Director: prof. Carlos Valderrama

2014 Petr Matas, "Material Architecture of an Intelligent Vision Sensor." Thesis in co-supervision with the University of West Bohemia in Pilsen, Czech Republic, defense planned for May 2014. Directors thesis: Mohamed Akil and Vjaceslav Georgiev, co-directors: Eva Dokladalova and Martin Poupa

2017 Ali Isavudeen "Study and design of dynamically adaptable on-chip network for embedded vision". Thesis CIFRE with SAGEM Defense, since 2014. Thesis supervisor: Mohamed Akil, co-directors: Eva Dokladalova, Nicolas Ngan

Undergoing thesis

2017 Elias Barbudo, “Adaptable processor for time critical image processing tasks”, director: L. George, co-director: Eva Dokladalova

2017 Rosemberg Osawaldo Rodrigez Salas, “New processing engine for embedded deep learning methods”, director: L. George, co-director: Eva Dokladalova, Petr Dokladal (Mines-ParisTech)

2017 Eric Bazan, “New perception methods for UAVs”, director: Petr Dokladal (Mines-ParisTech), co-director: Eva Dokladalova

Supervision of research internships

— Pavel Karas - PhD student at Masaryk University Brno, topic: Implantation of morphological filters on GPU

— Jorgé Tapia Hernandez - PhD student at Universidad Politecnica de Tulancingo, Mexico, topic : GPU implementation of unspervised targer detection for UAV

— Christophe Clienti - master research internship, CEA-LIST-LCEI, theme: Dedicated architecture for image processing

— Julien Krawczyk - SESI master course, UPMC, theme: SystemC modeling of dynamically adaptable NoC

— Amal Kachkach - SESI Master course, UPMC, theme: Multi-application self-adaptable architecture: implementation of the system monitor

— Dominik Schneider - master research internship, University of Pilsen, topic: Detection and classification of particles by TimePix detector

— Steeve Sivanantham, master research internship, University Paris-Est, theme: Deep learning: face recognition

— Giuliano Infantino, master research internship, University of Mons, Belgium, theme: Morphological tools for the classification of scenes.

— Antonie Smith, master research internship, Tshwane University of Technology (TUT), theme : Robust face detection under non controlled lighting conditions

— Alvaro Baquedano, research internship, University of Navarra, Spain, theme: Detection and tracking of target for automatic landing drones.

3.5 Technology Transfer and Industrial Projects

— *European project MEDEA +: CARVISION* - development of an innovative and reconfigurable embedded processor for embedded vision systems dedicated to automotive safety applications, 2 patents filed.

— *SAFRAN Defense - Centre of excellence for Optronics* - evolution of the hardware architecture for embedded vision part, product related to the thesis of Nicolas Ngan and Ali Isavudeen, with centre of excellence of optronics.

— *Target detection and recognition for automatic landing of drones* - in collaboration with the Internest company (start-up), we worked on the definition of a vision system for drones to facilitate the automatic landing of the drone.

3.6 National and international scientific collaborations

— **SAGEM Defense**: scientific collaboration on the theme of adaptable architectures for embedded vision.

— **Centre of Mathematical Morphology, Mines-ParisTech**: collaborations on the topic of algorithms for embedded vision and their hardware implementation.

— **University of West Bohemia in Pilsen, Czech Republic:**
- Faculty of Electrical Engineering: Scientific collaboration on the theme of embedded and adaptable architectures for visions. We have jointly supervised two co-supervised theses on this theme. Collaboration at the teaching level and reception of trainees and interns.
Initiative to set up a study program allowing students from ESIEE and the Czech University to obtain a double degree upon completion of this training. First two students graduated in 2014.
- Faculty of Applied Sciences: Participation in the setting up of a new ERASMUS convention for student exchange.

— **Technical University of Brno, Czech Republic:**
- Reception of research trainees
- Reception of researchers

— **Masaryk University in Brno, Czech Republic:**
- Scientific collaboration on accelerating GPU calculations in the past, NxP Cup preparation
- Initiative to create a new ERASMUS convention for the exchange of PhD students and students, today used by Master MOTIS at ESIEE Paris and MU

— **Institute of Applied and Experimental Physics of the Technical University of Prague, Czech Republic:**

- Scientific collaboration on the topic of embedded architectures, adapted to sensors of new generations with high dynamics and classification of particles.

— **University of Mons, Belgium:**

- Scientific collaboration, participation in the accompaniment of a doctoral student, exchange organization of Master students

- Initiative to create a new ERASMUS convention for the exchange of PhD students and students

— **University of Yucatan, Merida, Mexico:**

- Preparation of a collaboration agreement for the exchange of PhD students and students (in discussion)

- Research collaboration on the theme of non destructive analysis of composite materials (in discussion)

3.7 Grants

— Auto-adaptable computing support for military equipment (2 PhD thesis CIFRE), 60k€, SAGEM DEFENSE, France

— New adaptable processor for time critical image processing (PhD grant), 40k€, PhD thesis grant from CONACYT, Mexico

— New processing engine for embedded deep learning, 100k€, PhD thesis grant from UPE (3 years), France

3.7.1 Participation on research projects

I am involved in the preparation of the following projects :

— UrbaRiskLab, leader LATTS UMR 8134 CNRS, Ecole des Ponts, Université Paris-Est, I take in a charge MiniLab “Sensors and usage”

— DiXit, digital work sites, leader Laboratory Navier, ENPC, UPE, I coordinate the workpackage “Machine vision for construction sites”

3.8 Other activities and memberships

— Member of Computer Science Laboratory Gaspard Monge (LIGM) , CNRS

— Teaching involvement in Master of research, M2, *Science of the image* of Paris-Est University Marne la Vallée.

— Participation in:

GDR-ISIS, theme B and C

GDR-SOC / SIP

— Participation in the MEMICS Program Committee 2010, 2014, 2015, Czech Republic

— Program committee of the 1st and 2nd International Conference on Microelectronic Devices and Technologies (MicDAT’ 2018, MicDAT’2019)

— Program committee of the 24the International Conference on Applied Electronics, 2019

— Reviewer Service:

— Pattern Recognition Letters
— Journal of Visual Communication and Image Representation
— IEEE Transactions on Circuits and Systems for Video Technology
— Journal of Real-Time Image Processing
— IEEE Sensors journal
— Transactions on Parallel and Distributed Systems

Chapter 4

Institutional Responsibilities

4.1 Head of the Computer Science track (5th year)

Since September 2010 to February 2018, I was co-heading the Computer Science track. In principle, a track represents the last two years of engineering studies. It includes the steering of mandatory modules, optional modules and some other modules open to students from other tracks. The promotion of the sector and the motivation to choose it is also the responsability of its head.

Coordination of Computer Science Courses

> — Definition of teaching program : modules definition, general content
> — Placement of units in slots and coordination of the teachers availability
> — Creating new units, coordination of external spakers, mainly from private and industrial sector
> — Validation of final projects topics, coordination of the supervisors and defense organisation
> — Integration of external and foreing students
> — Management of particular student situations

To illustrate, I can cite some more concrete examples :

I participated in the creation of "career-oriented courses" in 2010. The following internal options have been created: General Computer Science, Software Engineering, Information Processing, Networks. In addition, three courses in collaboration with other institutions have been proposed: Bioinformatics (ESIEE-ISBS [1]), Computer Science and Geomatics (ESIEE-ENSG [2]), Multimedia (ESIEE-IMAC [3]). As such, I managed the selection and integration of ENSG students coming to follow lectures at ESIEE. In collaboration with V. Biri, I also managed the selection and participation of ESIEE students in teaching at IMAC.

In 2012, thanks to two new units I initiated we were able to open a new internal option " Multimedia, Virtual Reality, 3D" and "Infomatics for portable systems ". The interest of the students in this course motivated the creation of another unit, OpenGl, now given by Mustafa Nabil.

In 2014, I also participated in the transformation of the 5^{th} year of the computer science during the teaching reform at ESIEE. It is interesting to note that the computer science sector

1. ISBS = 'Institut Supérieur de BioSciences de Paris
2. École nationale des sciences géographiques
3. Ingénieur Image, Multimédia, Audiovisuel et Communication

is the one that offers, during this period, the students the largest choice of elective units.

During 4 years, I managed the partnership with MBDA [4] related to the IHM [5] unit which has made it possible to set up a patronage convention thanks to which ESIEE Paris has been receiving, over the last years, an average annual subsidy in terms of tens of k€.

In order to enlarge the competences of our students in the field of cyber security, I initiated the creation of the 'Security Audit 'unit (started at 2017), presented by an industrial partner DEVOTEAM, a main privately-held expert body in the field. This module has become a "Chair" last year.

Councils and boards

- Full time courses board
- Jury and progress assessment
- Diploma validation board
- Jury for the international departures
- Disciplinary board

Representation of the Computer Science Track

- Group of skils and competences, ESIEE engineer educational program evaluation
- Development Council
- Contact with companies, presentations of the cursus.

As a co-head of the Computer Science track, I contributed to writing of various documents such as the description of the skills acquired in the E5 units. I participated in the next Development Council of the institution. And in parallel, I participated in meetings with companies where I coordinated their teaching involvement in the track.

Presentations of the Computer Science track to students

- Presentations of the Computer Science track to students at the end of the third year of studies
- Presentations of the Computer Science track on the Education expositions, salons and others events.

I provided presentations in the computer science, first to help students choose their track and track, then to advise them in their choice of units in line with their track.

Also, I organized the presentation of the units contents to ensure a good understanding of the content by the students.

Other activities

- writing information about the course, objectives and evaluation.
- advice and approval of study programs abroad
- advice and help with research internship
- welcome students and/or parents when needed

4. MBDA Missile systems
5. french acronym for Man-Machine Interface

4.2 Participation in admissions

I participate in admission procedures:

— Oral exam for the Computer Networks track

— Oral exam of the classical track

— Evaluation of post-baccalaureate files

— Evaluation of N + I files (program for foreing students)

4.3 Participation in international collaborations

Since my arrival at ESIEE, I have set up collaborations with three leading Czech technical universities: the Technical University of Brno, the University of West Bohemia in Pilsen and the University of Masaryk in Brno.

In 2011, I created a **Double degree program** betwen ESIEE and the University of West Bohemia in Pilsen.

These collaborations made it possible to propose academic stays or students internships abroad. Reciprocally, in 2012-2013, two students from Pilsen were able to obtain the diploma of ESIEE as part of the double diploma. The scientific cooperation is as well established.

I participate in welcoming lecturers and researchers from these universities. I also organize the welcome of internship trainees.

I created a collaboration with the University of Mons, Belgium.

I created new collaboration with the University of Yucatán, Merida, Mexico.

For the past three years, I have been promoting Doctoral school MSTIC [6], ESIEE and Gaspard-Monge Computer Science Laboratory at **the Doctoral Tour in Mexico**, the aim of which is to promote French research, create new partnerships and obtain thesis grants to study in France. I participate in welcoming lecturers, researchers and internship trainees from these universities.

4.4 Correspondent for international projects

Since September 2018, I am one of three correspondents for international projects. I am in a charge of selected relationships with our international partners and of the coordination of the student projects with these institutions.

4.5 Participation in Open Days

I regularly contribute to the organization of the Open Days, either by ensuring the presence in a computer room arranged for the occasion for the reception of the public or by preparing demonstrations of the teachings.

This year, several demonstrations were provided and prepared by me:

— autonomous navigation of mobile robots, first year student project

— detection and rotation of text on industrial camera, third year student project

— real-time implementation of video stream processing, example of problems dealt with execution time optimisation.

— creation of a custom embedded Linux distribution (Yocto project).

— demonstration of industrial collaboration, SAGEM/Défense collaboration project

6. Mathematics and science of technology of information and communication

Chapter 5

Teaching activities

At present, I am associate professor, 2^{nd} degree, attached to the Computer Science Department of ESIEE Paris. My teaching activities cover four main areas :

1. Image processing
2. Embedded vision systems
3. Computer architecture and programing
4. Parallel implementations

I share my teaching activities in the First Cycle (1st-2nd year) and the Engineering Cycle (3rd-5th year) of ESIEE Paris.

I paticipate on the training by apprenticeship, in particular in Networks and Computer Science.

For several years, I participate in teaching in one of the international courses of the ESIEE, called *Computer Science Master*, taught entirely in English.

Also, I participate in the Research Master (M2) *Science of the image* where the ESIEE is cohabilitated with the Paris-Est University UPE.

5.1 Summary of some recent teaching activities

My current annual teaching load is 300 hours of equivalent practical work. A volume of 50 hours is dedicated to the management of the Computer Science track.

The following list presents a selection of units I have provided the last five years 2012-2017. For each unit, the theme associated with the interventions is specified as well as an example of the volume of hours insured in a reference year.

— Autonomous navigation of a mobile robot, 30h

— Introduction to science of an enginner, 24h

— Micropocessor programming, 50h

— Mobile systems and multimedia, 15h

— Optimisation for RISC processor, 40h (taught also in English)

— Multi-core optimisation, 8h

— Mobile and portable informatics, 30h

— Dedicated systems for VR/AR, 6h (Master research, UPE)

A new unit is being prepared with T. Grandpierre, it will be titled *Parallelism applied to data: finance and image*. It will focus on the introduction of OpenMP type tools, with real-time image processing domain applications or finance applications.

Notice that the microprocessor programming is based on the Texas Instruments robotics platforms and this teaching program has contributed to the development of large partnership and the opening of the first 4.0 library, TI Innovation Gateway (http://www.esiee.fr/en/opening-first-european-library-40-esiee-paris), managed by T. Grandpierre.

5.2 Other pedagogical activities

Systematically, I follow 5th, 4th and 3rd year projects.

Also, for years 2009-2012, I was the tutor of several apprentices in the Networks and security track. I followed apprentice students in big companies such as Orange, Bull, SOPRA, SAFRAN, THALES, UBISOFT, CEA but also SME companies like SOLUTEC, SPIDER-GAMES, and others.

With some projects, I participate in the competition for the best projects at ESIEE. Last year, a project I followed, won the Innovation Award. It was a smart and connected alarm clock, ensuring a qualitative follow-up of the sleep. (http://www.nozz.fr/)

5.3 Pedagogical methods

In this part, I highlight some the pedagogical approaches I implement

Active Pedagogy in Undergraduate and Engineering Cycle

- Pedagogy by project
- Exercise learning
- Educational Coaching
- Contest of the best website of the project
- Multidisciplinary and personalized team project
- Project management component for each completed project
- Self-Assessment QCMs
- Customization of completed projects / TPs
- Experimentation of the inverted class (planned for 2017/2018)

Evaluation of teaching

At ESIEE Paris, to improve the quality of teaching, each unit is subject to an evaluation by the students. Evaluations of my units can be made available upon request.

Chapitre 6

Liste des publications

Journaux internationaux

[1] J. BARTOVSKÝ, P. DOKLÁDAL, M. FAESSEL, E. DOKLADALOVA et M. BILODEAU. "Morphological co-processing unit for embedded devices". In : *Journal of Real-Time Image Processing* (2015), p. 1-12.

[2] J. BARTOVSKÝ, P. DOKLÁDAL, E. DOKLADALOVA et V. GEORGIEV. "Parallel implementation of sequential morphological filters". In : *Journal of Real-Time Image Processing* 9.2 (2014), p. 315-327.

[3] J. BARTOVSKÝ, P. DOKLÁDAL, E. DOKLADALOVA, M. BILODEAU et M. AKIL. "Real-time implementation of morphological filters with polygonal structuring elements". In : *Journal of Real-Time Image Processing* 10.1 (2015), p. 175-187.

[4] E. DEJNOZKOVA et P. DOKLADAL. "A parallel architecture for curve-evolution PDEs". In : *Journal of Image Analysis and Stereology* 22.2 (2003), p. 121-132.

[5] E. DEJNOZKOVA et P. DOKLADAL. "Embedded real-time architecture for level- set-based active contours". In : *EURASIP Journal on Advances in Signal Processing* 17.1 (2005), p. 1-16.

[6] P. DOKLÁDAL et E. DOKLADALOVA. "Computationally efficient, one-pass algorithm for morphological filters". In : *Journal of Visual Communication and Image Representation* 22.5 (2011), p. 411-420.

[7] P. KARAS, V. MORARD, J. BARTOVSKÝ, T. GRANDPIERRE, E. DOKLADALOVA, P. MATULA et P. DOKLADAL. "GPU implementation of linear morphological openings with arbitrary angle". In : *Journal of Real-Time Image Processing* 10.1 (2015), p. 27-41.

[8] N. NGAN, E. DOKLADALOVA, M. AKIL et F. CONTOU-CARRERE. "Fast and efficient FPGA implementation of connected operators". In : *Journal of Systems Architecture* 57.8 (2011), p. 778-789.

[9] P. POSSA, N. HARB, E. DOKLADALOVA et C. VALDERRAMA. "P2IP: A novel low-latency Programmable Pipeline Image Processor". In : *Microprocessors and Microsystems* 39.7 (2015), p. 529-540.

Chapitres en LNCS

[10] Eric BAZAN, Petr DOKLADAL et Eva DOKLADALOVA. "Non supervised perceptual model for target recognition in UAVs". In : *International Conference on Advanced Concepts for Intelligent Vision Systems*. Poitier, France, 2018.

[11] L. BIANCARDINI, E. DOKLADALOVA, S. BEUCHER et L. LETELLIER. “From moving edges to moving regions”. In : *Iberian Conference on Pattern Recognition and Image Analysis*. 2005, p. 119-127.

[12] E. DEJNOZKOVA et P. DOKLÁDAL. “Modelling of overlapping circular objects based on level set approach”. In : *LNCS : IMAGE ANALYSIS AND RECOGNITION (International Conference on Image Analysis and Recognition (ICIAR'04))*. T. 3211. 1. 2004, p. 416-423.

[13] P. DOKLADAL et E. DOKLADALOVA. “Grey-scale morphology with spatially-variant rectangles in linear time”. In : *Advanced Concepts for Intelligent Vision Systems*. 2008, p. 674-685.

[14] A. ISAVUDEEN, E. DOKLADALOVA, N. NGAN et al. “Self-Adaptive Architecture for Multi-sensor Embedded Vision System”. In : *LNCS*. T. 9548. Springer, 2015.

[15] P. MATAS, E. DOKLADALOVA, M. AKIL et al. “Parallel algorithm for concurrent computation of connected component tree”. In : *International Conference on Advanced Concepts for Intelligent Vision Systems*. 2008, 230-241.

Conférences internationales avec commité de lecture

[16] E BARBUDO, Eva DOKLADALOVA, Th GRANDPIERRE et L GEORGE. “A Mapping Methodology for Coarse-Grained Pipelined Configurable Architectures”. In : *14th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2019)*. Renesse, Netherlands, juin 2019.

[17] Elias BARBUDO, Eva DOKLADALOVA, Thierry GRANDPIERRE et Laurent GEORGE. “A New Mapping Methodology for Coarse-Grained Programmable Systolic Architectures”. In : *22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES 2019)*. St Goar, Germany, mai 2019.

[18] J. BARTOVSKÝ, E. DOKLADALOVA, P. DOKLÁDAL et M. AKIL. “Efficient FPGA architecture for oriented 1-D opening and pattern spectrum”. In : *Image Processing (ICIP), 2012 19th IEEE International Conference on*. IEEE. 2012, p. 1689-1692.

[19] J. BARTOVSKÝ, P. DOKLADAL, E. DOKLADALOVA et M. BILODEAU. “Fast streaming algorithm for 1-D morphological opening and closing on 2-D support”. In : *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. Springer Berlin Heidelberg. 2011, p. 296-305.

[20] J. BARTOVSKÝ, D. SCHNEIDER, E. DOKLADALOVA, P. DOKLÁDAL, V. GEORGIEV et M. AKIL. “Morphological classification of particles recorded by the timepix detector”. In : *Image and Signal Processing and Analysis (ISPA), 2011 7th International Symposium on*. IEEE. 2011, p. 343-348.

[21] J. BARTOVSKÝ, P. DOKLÁDAL, E. DOKLADALOVA et M. BILODEAU. “One-scan algorithm for arbitrarily oriented 1-D morphological opening and slope pattern spectrum”. In : *Image Processing (ICIP), 2012 19th IEEE International Conference on*. IEEE. 2012, p. 133-136.

[22] J. BARTOVSKÝ, E. DOKLADALOVA, P. DOKLÁDAL et V. GEORGIEV. “Pipeline architecture for compound morphological operators”. In : *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE. 2010, p. 3765-3768.

[23] J. BARTOVSKÝ, P. DOKLADAL, E. DOKLADALOVA et V. GEORGIEV. “Stream implementation of serial morphological filters with approximated polygons”. In : *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*. IEEE. 2010, p. 706-709.

[24] M. BENKINIOUAR, M. AKIL, E. DOKLADALOVA et M. BENMOHAMMED. “Exploration of VLIW space processor for adaboost-based applications”. In : *Applications of Digital Information and Web Technologies, 2009. ICADIWT'09. Second International Conference on the*. IEEE. 2009, p. 664-671.

[25] M. BENKINOUAR, M. AKIL, E. DOKLADALOVA et M. BENMOHAMED. “Détection des visages: étude algorithmique et implantation temps réel”. In : *4ème AMINA, Applications Médicales de l'Informatique: Nouvelles Approches*. 2008.

[26] E. DEJNOZKOVA et P. DOKLADAL. “A multiprocessor architecture for PDE based applications”. In : *International Conference on Visual Information Engineering (VIE 2003). Ideas, Applications, Experience, Guildford UK*. 2003.

[27] E. DEJNOZKOVA et P. DOKLADAL. “A parallel algorithm for solving the Eikonal equation”. In : *IEEE Intrernational conference on acoustics, speech and signal processing ICASSP*. 2003.

[28] E. DEJNOZKOVA et P. DOKLADAL. “Asynchronous multi-core architecture for level set methods”. In : *IEEE International Conference on Accoustics, Speech and Signal Pro- cessing (ICASSP'04)*. T. 5. 2004.

[29] E. DEJNOZKOVA, P. DOKLADAL et J.C. KLEIN. “Massive marching: a parallel computation of distance”. In : *IWSSIP, Recent Trends in Multimedia Information Processing*. World Scientific. 2002, p. 71-76.

[30] P. DOKLADAL et E. DOKLADALOVA. “Fast implementation of variational, contour-based object tracking”. In : *Signal Processing Conference, 2005 13th European*. IEEE. 2005, p. 1-4.

[31] P. DOKLADAL, R. ENFICIAUD et E. DEJNOZKOVA. “Contour-based object tracking with gradient-based contour attraction Field.” In : *IEEE International Conference on Accoustics, Speech and Signal Processing (ICASSP'04)*. T. 3. 17-20. 2004.

[32] Petr DOKLÁDAL et Eva DOKLADALOVA. “Grey-scale 1-D dilations with spatially-variant structuring elements in linear time”. In : *Signal Processing Conference, 2008 16th European*. IEEE. 2008, p. 1-5.

[33] E. DOKLADALOVA, S. CHEVOBBE et F. BLANC. “Advances in Practical Implementation of the Digital Media Processing: Towards Reconfigurable Computation”. In : *European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology (EWIMT'04)*. 1. 2004, 10pp.

[34] Eva DOKLADALOVA, R SCHMIT, S PAJANIRADJA et S AMADORI. “Carvision: SOC architecture for dynamic vision systems from image capture to high level image processing”. In : *MEDEA DAC*. 1. 2006, 10pp.

[35] Ph. GUERMEUR, P. DOKLÁDAL, E. DOKLADALOVA et A. MANZANERA. “FPGA lab sessions in a general-purpose image processing course”. In : *2nd International Workshop on Reconfigurable Computing Education*. 1. 2007, 10pp.

[36] A. ISAVUDEEN, N. NGAN, E. DOKLADALOVA et al. “Auto-Adaptive Multi-Sensor Architecture”. In : *IEEE ISCAS 2016*. 2016.

[37] A. ISAVUDEEN, N. NGAN, E. DOKLADALOVA et M. AKIL. “Highly Scalable Monitoring System on Chip for Multi-Stream Auto-Adaptable Vision System”. In : *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. RACS ’17. Krakow, Poland : ACM, 2017, p. 249-254. ISBN : 978-1-4503-5027-3.

[38] J. KLEIN, T. WALTER, P. MASSIN et E. DEJNOŽKOVÁ. “Computer aided diagnosis of diabetic retinopathy: A survey”. In : *Computer Assisted Fundus Image Analysis*. 2003.

[39] P. MATAS, E. DOKLADALOVA, M. AKIL, V. GEORGIEV et M. POUPA. “Parallel hardware implementation of connected component tree computation”. In : *Field Programmable Logic and Applications (FPL), 2010 International Conference on*. IEEE. 2010, p. 64-69.

[40] N. NGAN, E. DOKLADALOVA et M. AKIL. “Dynamically adaptable NoC router architecture for multiple pixel streams applications”. In : *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*. IEEE. 2012, p. 1006-1009.

[41] N. NGAN, E. DOKLADALOVA, M. AKIL et F. CONTOU-CARRERE. “Dynamically adaptable architecture for real-time video processing”. In : *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE. 2010, p. 4125-4128.

[42] N. NGAN, F. CONTOU-CARRÈRE, B. MARCON, S. GUERIN, E. DOKLADALOVA et M. AKIL. “Efficient hardware implementation of connected component tree algorithm”. In : *Workshop on Design and Architectures For Signal and Image Processing*. 1. 2007, 10pp.

[43] N. NGAN, G. MARPEAUX, E. DOKLADALOVA, M. AKIL et F. CONTOU-CARRERE. “Memory system for a dynamically adaptable pixel stream architecture”. In : *Field Programmable Logic and Applications (FPL), 2010 International Conference on*. IEEE. 2010, p. 199-204.

Conférences et colloques nationaux

[44] E. BARBUDO, E. DOKLADALOVA, T. GRANDPIERRE et L. GEORGE. “A mapping tool for configurable pipeline co-processors”. In : *Colloque GDR-SOC*. Paris, France, juin 2018.

[45] E. BAZAN, P. DOKLADAL et E. DOKLADALOVA. “Non supervised perceptual model for target recognition in UAVs”. In : *Reconnaissance des Formes, Image, Apprentissage et Perception RFIAP*. Marne la Vallée, France, juin 2018.

[46] A. ISAVUDEEN, E. DOKLADALOVA, N. NGAN et M. AKIL. “FPGA-based Architecture for Smart Multi-Sensor Embedded Vision System”. In : *5th Workshop on the Architecture of Smart Cameras, Dijon, France*. Juil. 2016.

Séminaires scientifiques

[47] E. DOKLADALOVA. “Computationally efficient algorithms for mathematical morphology”. Universidad Autonoma Metropolitana, Mexico City, Mexique. 2017.

[48] E. DOKLADALOVA. “Efficient Hardware Architectures and Algorithms for Embedded Vision Systems”. Masaryk University, Brno, Czech Republic. Oct. 2015.

[49] E. DOKLADALOVA et M. AKIL. “Imagerie temps réel et Architecture : Architectures pour l’imagerie : défis et perspectives”. École STIC 2013, Monastir, Tunisie. 2013.

[50] Eva DOKLADALOVA. “Architecture matérielle parallèle, dédiée et programmable pour les filtres morphologiques avec des grands éléments structurants”. École STIC 2013, Monastir, Tunisie. 2013.

[51] T. Grandpierre et E. Dokladalova M. AKIL. “Cartes à puce, architectures et protocoles”. Séminaire, d’école d’été, INPT, Rabat, Maroc. 2007.

Rapports

[52] J. BARTOVSKÝ, Eva DOKLADALOVA, Michel BILODEAU et Petr DOKLÁDAL. *Texture Analysis with Arbitrarily Oriented Morphological Opening and Closing*. Rapp. tech. 2011.

[53] E. DEJNOZKOVA. *Architecture dédiée au traitement d’image basé sur les équations aux dérivées partielles*. Rapp. tech. PhD thesis, CMM ENSMP, https://tel.archives-ouvertes.fr/pastel-00001180/, 2004.

[54] E. DEJNOZKOVA, P. DOKLADAL et J.C. KLEIN. *Massive marching: A parallel computation of distance function for PDE-based applications*. N-17/02/MM. 2002.

[55] E. DOKLADALOVA. *Etude d’implémentation matérielle de l’encodeur MPEG4-AVCH.264*. Rapp. tech. Rapport de stage post-doctoral, CEA-LIST-LCEI, Saclay. 2005.

Brevets

[56] E. DOKLADALOVA, Ph. FAUVEL, S. GUYETANT et Ch. CLIENTI. *Image processing system with morphological macro cell*. US Patent US 20100142855 A1. 2007.

[57] S. PAJANIRADJA, E. DOKLADALOVA, M. GUIBERT et M. ZEMB. *Device with datastream pipeline architecture for recognizing and locating objects in an image by detection window scanning*. US Patent App. 13/133,617. 2009.

En préparation

Journaux internationaux

[58] RODRIGUEZ SALAS, ROSEMBERG AND DOKLÁDAL, PETR AND DOKLADALOVA, EVA. “Adaptive steerable filter equivariant convolution network for classification and automatic reorientation of angle blind datasets”. In : *IEEE Transactions on Neural Networks and Learning Systems* (2019). in preparation, p. -.

Conférences internationales

[59] E BARBUDO, Eva DOKLADALOVA, Th GRANDPIERRE et L GEORGE. “Complete mapping framework for Coarse-Grained Pipelined Configurable Architectures”. In : *DASIP 2019*. Montréal, Canada, juin 2019.

[60] Eric BAZAN, Petr DOKLÁDAL et Eva DOKLADALOVA. “Quantitative Analysis of Similarity Measures of Distributions”. In : *BMVC 2019,* in preparation for submission. 2019.

[61] RODRIGUEZ SALAS, ROSEMBERG AND DOKLÁDAL, PETR AND DOKLADALOVA, EVA. “Rotation-invariant steerable convolution for image classification”. In : *BMVC 2019,* in preparation for submission. 2019.

[62] RODRIGUEZ SALAS, ROSEMBERG AND DOKLADALOVA, EVA AND DOKLÁDAL, PETR. “Rotation invariant CNN using scattering transform for image classification”. In : *ICIP 2019,* submitted. Fév. 2019.

Mémoire de HDR

[63] E. DOKLADALOVA. “Algorithmes et architectures efficaces pour vision embarquée”. In : *Université Paris-Est* (2018). mémoire de Habilitation à Diriger des Recherches.

Deuxième partie

Contributions algorithmiques

Chapitre 7

Cartes de distance à précision sous-pixellique

Le calcul des cartes de distance à précision sous-pixelique est basé sur des techniques des ensembles de niveaux (*level set*). La popularité de ces techniques est due à Sethian et Osher qui ont démontré que les techniques des ensembles de niveaux permettent de contrôler efficacement des changements topologiques et de simplifier les problèmes numériques [147].

Le calcul repose sur la propagation d'une frontière représentée par une courbe fermée et par une fonction F indiquant la vitesse de propagation de courbe dans la direction normale. F est également appelée la fonction de coûts (poids) de propagation. La frontière de propagation est souvent appelée Γ et se propage vers l'intérieur ou l'extérieur de la région, en fonction du signe de F. À chaque passage d'un pixel, une valeur de la distance proportionnelle au temps de passage [72] lui est attribuée.

Pour illustrer, considérons la fonction $\phi : \mathbb{Z} \to \mathbb{R}$ initialisée de façon $\phi(x, t = 0) = \pm d$, où d représente la distance de $x \in \mathbb{Z}^2$ à la courbe initiale Γ. Le signe plus (ou moins) est donné selon si la position de x est à l'extérieur (ou à l'intérieur) de Γ. La courbe Γ est le niveau zéro de la fonction $\phi(x, t = 0)$. Le front de propagation se déplace avec la vitesse $F = F(x, y, z), F > 0$.

Ainsi l'évolution de la frontière Γ est donnée par l'équation suivante :

$$\phi_t + F\|\nabla\phi\| = 0 \tag{7.1}$$

$$\phi(x, t = 0) = \Gamma \tag{7.2}$$

Si on représente l'évolution de l'ensemble du niveau 0 sur un plan 2D, le problème devient stationnaire :

$$F\|\nabla\phi\| = 1 \tag{7.3}$$

Il s'agit de la célèbre équation eikonale qui est commune à de nombreuses applications dont on peut citer quelques-unes : reconstruction de la surface 3D [172], les plus courts chemins [167], segmentation [66], la distance géodésique [164].

7.1 Equation eikonale

L'algorithme le plus souvent employé pour calculer la propagation du front selon l'équation eikonale (Eq. 7.3) est l'algorithme **Fast Marching** [158], basé sur l'algorithme de Dijkstra.

Dans la pratique, le principe repose sur une propagation équidistante de la solution numérique de l'équation eikonale. Pour cela, on traite les pixels autour du front de propagation dans un ordre de priorité. La priorité maximale est donnée aux pixels dont la valeur représente la distance minimale à l'instant de passage du front.

L'inconvénient évident est que cela introduit le besoin de l'appréciation globale des priorités et rend cette approche quasi-séquentielle, dépendant de la structure de données utilisée pour le tri des priorités. Les implantations pratiques reposent sur les structures de données comme le tas de Fibonacci qui permettent d'amortir le coût des opérations sur des éléments du tas, mais il peut, dans le pire cas, atteindre des valeurs élevées telles que $O(n)$ [102] et pénaliser ainsi les applications nécessitant le temps réel.

Une approche partiellement parallèle est proposée dans l'algorithme **Group Marching** [120]. Cependant, l'algorithme nécessite d'évaluer un critère global pour sélectionner un groupe de points, à priorité égale, qui seront traités dans une itération dans un ordre défini.

7.2 Massive Marching : algorithme massivement parallèle à propagation asynchrone

Dans les travaux issus de ma thèse [53], je propose un algorithme parallèle appelé **Massive Marching**. Il repose sur un mécanisme d'une propagation non équidistante et asynchrone dotant cette approche de haut potentiel de parallélisation tout en préservant la qualité de solution numérique. Nous présentons ci-dessous ces différentes étapes.

7.2.1 Initialisation

Pour identifier la courbe initiale, nous adoptons l'approche par interpolation proposée par Sethian [129] et qui est illustrée sur la Fig. 7.1. L'algorithme reprend la technique de l'interpolation linéaire [161] permettant d'affecter des valeurs connues pré-calculée $\phi : \mathbb{Z}^2 \rightarrow \{c_1, c_2, ..., c_n\}$ où $c_i \in \mathbb{R}$ à tous les pixels voisins du front initial.

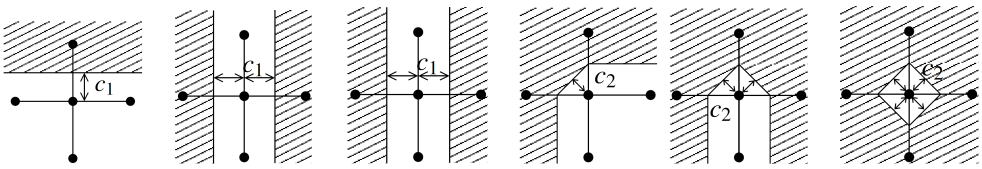


FIGURE 7.1 – Interpolation linéaire : différentes configurations des points voisins.

À l'issue de l'étape d'interpolation nous connaissons la position de la courbe initiale. Tous les autres points sont initialisés à l'infini.

7.2.2 Schéma numérique

Le choix du schéma numérique est orienté par sa précision, par sa stabilité et par sa complexité de calculs. Le schéma le plus largement utilisé est le suivant :

$$[max(max(D_{ij}^{-x}\phi,0),-min(D_{ij}^{+x}\phi,0))^2+max(max(D_{ij}^{-y}\phi,0),-min(D_{ij}^{+y}\phi,0))^2] = \frac{1}{{F^2}_{ij}} \tag{7.4}$$

où $D_{ij}^{\pm x}$, $D_{ij}^{\pm y}$ représentent les dérivées partielles numériques de gauche ($-x$) et de droite ($+x$) selon x (ou y), par exemple : $D_{ij}^{+x}\phi = (\phi_{i+1,j} - \phi_{i,j})/(\Delta x)$.

Ce schéma permet de garantir la solution de viscosité présentée en [147]. En effet, il s'agit de résoudre une équation quadratique, qui est résolue pour chaque point de la grille en supposant que les autres points sont fixes.

7.2.3 Les principes de Massive Marching

Considérons le point $p = [x_p, y_p]$, $p \in \mathbb{Z}^2$ sur une grille rectangulaire et isotrope. Nous appelons $V(p)$ le 4-voisinage du point p, ainsi $V(p) = \{[x_p, y_p \pm 1], [x_p \pm 1, y_p]\}$. Ainsi le point q est un voisin de p si $q \in V(p)$. $\phi(p)$ représente la valeur de la fonction distance en point p. Pour simplifier, nous supposons que le pas de discrétisation $h = 1$.

Pour définir l'algorithme, nous utilisons les ensembles suivants : $\mathscr{Q}$ est l'ensemble de points initialisés par l'interpolation, $\mathscr{A}$ est l'ensemble de points marqués comme actifs obtenus par $\mathscr{A} = \{q_i \mid q_i \notin \mathscr{Q} \text{ et } V_4(q_i) \cap \mathscr{Q} \neq \emptyset\}$.

Algorithm 1: Massive Marching (MM) [53]

Data: $\mathscr{A}, u_0, \mathscr{F}, \mathscr{Q}$
Result: ϕ
begin

Tant que $\mathscr{A} \neq \{\}$, faire **en parallèle** pour tous les points actifs ($p \in \mathscr{A}$) :
{

⋆ *Calculer la nouvelle valeur de distance* :

- **Itération de Jacobi :**

$$\phi^{n+1}(p) = \phi^{n}_{min}(p) + \min\left\{f_{diff}\left(V_4(p), \mathscr{F}(p)\right), \mathscr{F}(p)\right\} \tag{7.5}$$

- **Itération de Gauss-Seidel :**

$$\phi^{n+1}(p) = \phi^{n+1}_{min}(p) + \min\left\{f_{diff}\left(V_4(p), \mathscr{F}(p)\right), \mathscr{F}(p)\right\} \tag{7.6}$$

⋆ *Activer des nouveaux points pour l'itération suivante :*

- Effacer p de $\mathscr{A}$ et insérer p en $\mathscr{Q}$.
- Si $\phi(p) < \mathrm{NB}_{\mathrm{width}}$ alors $\forall\, q_i,\ q_i \in V_4(p)$ tel que :

$$\phi^{n+1}(q_i) - \phi^{n+1}(p) > \varepsilon(q_i) \tag{7.7}$$

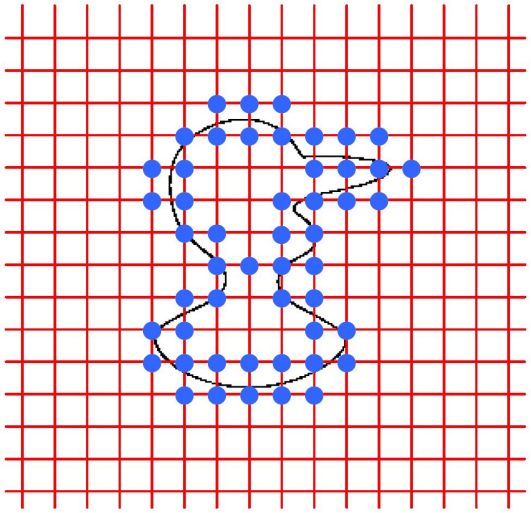
insérer $q_i \rightarrow \mathscr{A}$

}

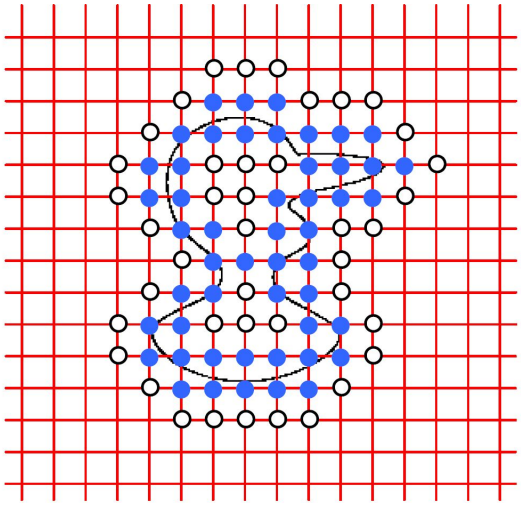
où $\mathrm{NB}_{\mathrm{width}}$ est la demi largeur choisie de la bande sur laquelle le résultat est calculé (Narrow Band). En mettant $\mathrm{NB}_{\mathrm{width}} = \infty$, nous pouvons calculer la carte de distance sur l'ensemble des points de l'image.
Avec

$$f_{diff} = \frac{\phi_x(p) - \phi_y(p)}{2} + \sqrt{\frac{\mathscr{F}(p)}{2} - \left(\frac{\phi_x(p) - \phi_y(p)}{2}\right)^2} \tag{7.8}$$

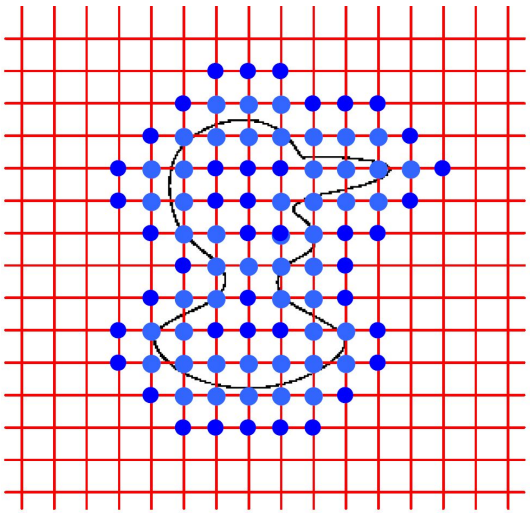
$\phi_x(p) = min\{\phi(x_p \pm 1, y_p)\}$ et $\phi_y(p) = min\{\phi(x_p, y_p \pm 1)\}$ $\mathscr{F}(p)$ est la fonction de coût (de vitesse) de propagation.



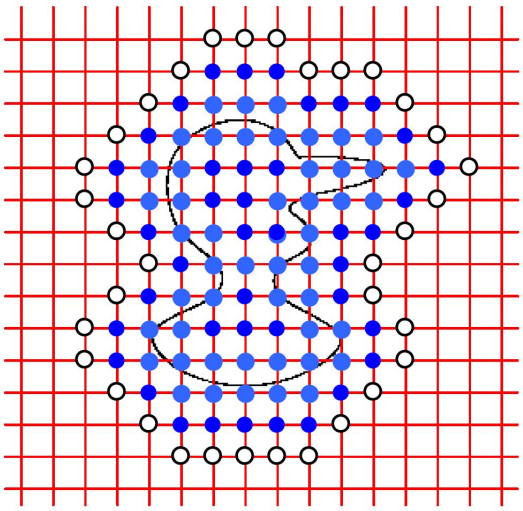
(a) Détection du contour : les voisins les plus proches du contour sont initialisés par l'interpolation (points bleus).



(b) Les pixels dont les valeurs seront calculés pendant la première itération sont marqués comme *actifs*, appartenant à l'ensemble $\mathscr{A}$. (points blancs).



(c) Après les deux étapes de calcul Jacobi et Gauss-Seidel, tous les points actifs sont désactivés, points déjà traités appartiennent à l'ensemble $\mathscr{Q}$.



(d) Les voisins des points, qui viennent d'être traités, sont marqués comme *actifs* s'ils satisfont le critère d'activation.

FIGURE 7.2 – Première itération de l'algorithme.

Deux itérations internes

Lors de la propagation asynchrone, la valeur d'un point peut être calculée sur un voisinage numériquement incomplet. C'est-à-dire, les points adjacents (réciproquement dépendants) peuvent être traités simultanément même si leurs distances ne sont pas égales. C'est la raison d'employer deux itérations, conforme à l'algorithme d'approximation de chaîne de Markov par EDPs [1] [76].

La première itération, *itération de Jacobi*, calcule la valeur de la fonction de la distance à t_{n+1} en fonction des valeurs obtenues à t_n. La seconde itération, *itération de Gauss-Seidel*, précise la valeur de distance à t_{n+1} en utilisant aussi les valeurs obtenues à t_{n+1}. Notons que l'algorithme ne calcule la valeur $\phi(p)$ qu'à partir du voisin minimal (Eq. (7.8)).

1. EDPs – Équations aux Dérivées Partielles

Critère d'activation

Contrôler la fin de la propagation et permettre de corriger erreur en cas d'une propagation "trop rapide" est le rôle du critère d'activation. Il doit garantir l'obtention de la solution minimale de la fonction distance. Nous définissions un estimateur de ϕ^{n+1} pour savoir quels voisins q_i marquer comme actifs pour la prochaine itération.

La définition du critère d'activation repose donc les propriétés lipschitziennes de la fonction distance euclidienne. Tout segment reliant deux points du graphe de la fonction distance aura une pente inférieure, en valeur absolue, à une constante.

On sait que chaque nouvelle valeur $\phi(q_i)$ satisferait $\phi(q_i) \geq u(p) + \inf f_{diff}$.

Soit K_{min}la borne inférieure de f_{diff} :

$$K_{\min}(p) = \inf f_{diff}(V_4(p), \mathscr{F}(p)) \tag{7.9}$$

$\mathscr{F}(p)$ est une fonction arbitraire mais invariante dans le temps. $K_{\min}$ est le prédicteur du plus petit incrément de ϕ dans une itération. Tous les voisins q_i de p tels que $\phi(q_i) - \phi(p) > K_{\min}(p)$ devraient donc être activés à nouveau et recalculés car la nouvelle valeur $\phi(p)$ pourrait affecter $\phi(q_i)$ dans l'itération suivante. Les détails sont présentés en [4]. La borne inférieure de f_{diff} du schéma Godunov est

$$K_{\min}(p) = \mathscr{F}(p)\frac{1}{\sqrt{2}} \tag{7.10}$$

K_{min}est constant si $\mathscr{F}$ est constante et devient une fonction de $\mathscr{F}$ si $\mathscr{F}$ varie sur l'image.

7.2.4 Propagation des étiquettes

La propagation des étiquettes des régions peut être exécutée simultanément avec le calcul de la fonction de distance pour obtenir les zones d'influence du diagramme de Voronoï ou de la ligne de partage des eaux continue [143].

La Fig. 7.3 montre comment Massive Marching procède pour calculer la fonction distance pondérée à partir des sources. Nous pouvons y observer comment la valeur de la fonction distance est propagée sur l'image entière et comment les contours (et des valeurs) déjà calculés, sont recalculés grâce au dépassement des différentes ondes.

Nous pouvons observer le principe du dépassement des ondes avec des vitesses différentes. D'abord, les ondes envahissent l'espace de l'image en fonction de la définition du voisinage utilisé pour le schéma numérique. À la rencontre de deux ou plusieurs ondes, les plus "rapides" s'arrêtent et les plus "lentes" continuent à propager les étiquettes.

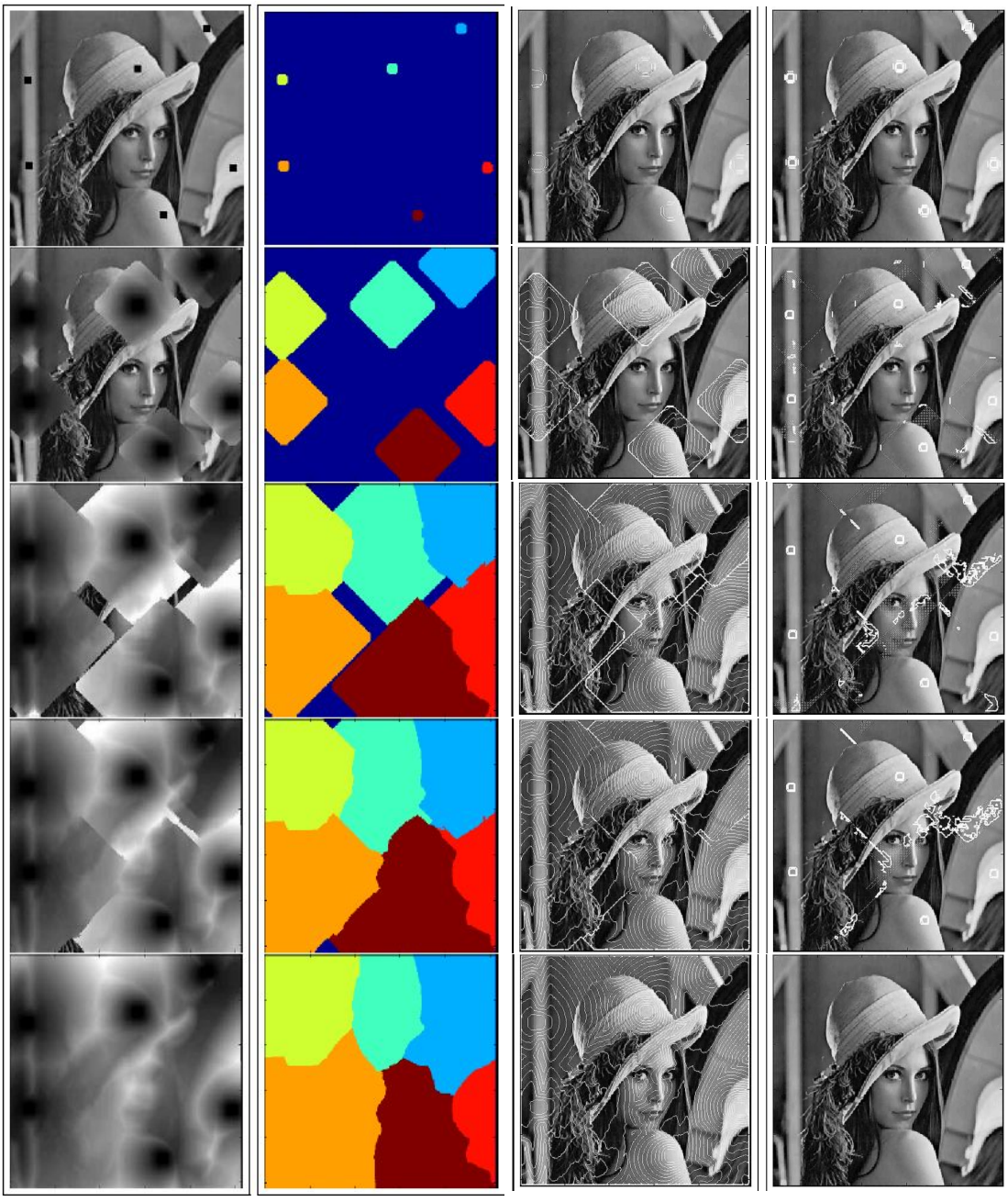


FIGURE 7.3 – Propagation de la fonction distance. De gauche à droite : image initiale avec les sources de propagation et la fonction distance pondérée par des valeurs d'intensité de l'image, les étiquettes propagées en même temps, contours de la fonction distance, points actifs (ensemble $\mathscr{A}$) dans l'itération en cours.

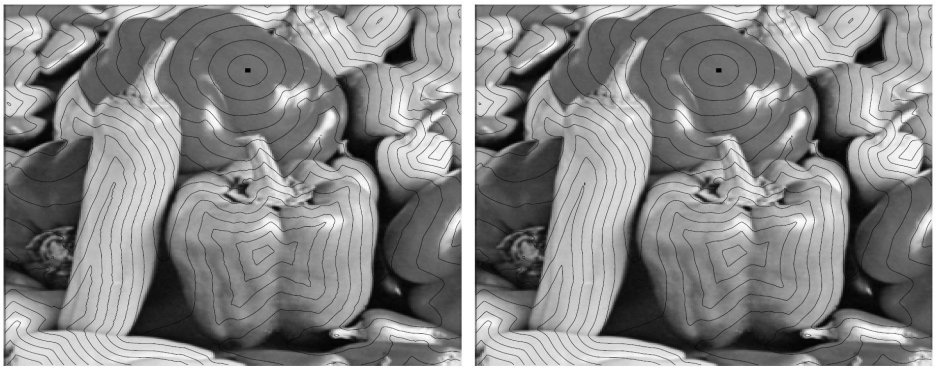


FIGURE 7.4 – Comparaison de deux différents algorithmes du calcul de distance pondérée. A gauche : Massive Marching, à droite : propagation équidistante

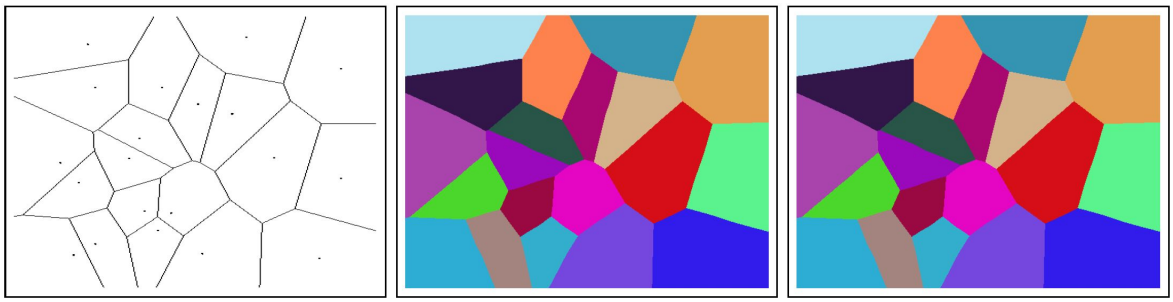


FIGURE 7.5 – Diagramme de Voronoï obtenu par : à gauche - la triangulation de Delaunay ; au centre - Massive Marching ; à droite - propagation équidistante.

7.3 Évaluation

Erreur de voisinage incomplet

Massive Marching calcule en parallèle les valeurs de points adjacents. Les valeurs des points adjacents peuvent s'influencer mutuellement. Ainsi, pour assurer la cohérence des résultats, le calcul est exécuté en deux étapes : Jacobi et Gauss-Seidel.

Les Fig. 7.4 et Fig. 7.5 montre la comparaison entre les résultats obtenus à l'aide de Massive Marching et une autre méthode concurrente.

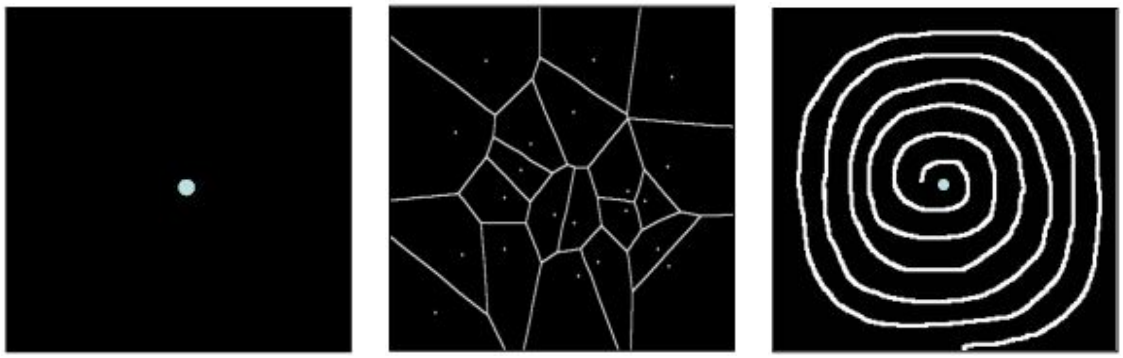
Complexité de calcul

Quelle que soit la vitesse de propagation $\mathscr{F}$, la valeur d'un pixel actif p est obtenue dans un temps constant $\mathscr{O}(1)$ après quoi le pixel lui-même devient inactif. La fonction (7.8) est strictement positive, aucun pixel ne peut réactiver le voisin duquel il a reçu l'activation. Par conséquent, la complexité de calcul est linéaire $\mathscr{O}(N)$ de N est le nombre de pixels à traiter.

Nombre d'opérations

Si la propagation est faite à partir d'une seule source (Fig. 7.6(a)), le nombre d'opérations atteint la limite inférieure $O(N)$ donnée par la complexité de calcul de l'algorithme.

Pour la propagation à partir de plusieurs sources (Fig. 7.6(b)) ou des sources ayant des formes géométriques plus compliquées, la complexité peut dépasser $O(N)$ parce que plusieurs points peuvent être activés plus d'une fois à cause des dépassements. Il peut être démontré que le nombre de pixels réactivés est minimal, le nombre d'opérations est estimé à $O(N+k_1)$, k_1 étant une constante indépendante de N.



(a) Une source unique et $\mathscr{F} = const.$ (b) Plusieurs sources et $\mathscr{F} = const.$ (c) Une (ou plusieurs) source(s) avec $\mathscr{F} \neq const.$

FIGURE 7.6 – Illustration des différentes configurations des sources de propagation.

Pour les cas où $\mathscr{F} \neq const.$, l'estimation du nombre d'opérations est très complexe car dépendant de $\mathscr{F}$, du nombre de sources et de leur forme en même temps. Dans le cas extrême, présenté sur la Fig. 7.6(c), le nombre de points recalculés est élevé parce que la réactivation des points peut engendrer la propagation du front sur une grande partie de l'image. Par conséquent, le nombre d'opérations peut être estimé à $O(k_2 N)$ où k_2 est une constante indépendante de N et donc bornée. Notons que les expériences effectuées ont toutefois montré que le temps de traitement est toujours de même ordre de grandeur par rapport d'autres algorithmes [53].

La pertinence de la solution est démontrée par une grande multiplicité des publications qui adressent cette problématique.

En 2005, Zhao publie en [180] une méthode à complexité réduite $\mathcal{O}(N)$ appelée *Fast sweeping* qui selon les comparaisons publiées en [80] est plus efficace en séquentiel que les méthodes classiques basées sur Fast marching. En 2006, les auteurs de [178] proposent un algorithme à complexité linéaire $\mathcal{O}(N)$ pour la solution de l'équation eikonale, la réduction de la complexité de calcul est obtenue grâce à une structure de donnée basée sur la quantification des priorités des points à traiter. En 2008, Weber et al. [173] proposent également une autre implantation de la résolution de l'équation eikonale avec une complexité $\mathcal{O}(N)$ pouvant être efficacement implantée sur les GPUs. Les auteurs optent pour la suppression des structures de données ordonnées et proposent un calcul à balayage régulier des points à calculer. Pour cela, ils mettent en œuvre la vérification de l'erreur du calcul qui est similaire au critère d'activation du Massive Marching.

Chapitre 8

Opérateurs morphologiques de grande taille, à flot de données

Dans ce chapitre nous présentons des contributions dans le domaine des algorithmes efficaces pour les opérateurs morphologiques concaténés.

Les algorithmes proposés ont une latence d'algorithme nulle et un accès strictement séquentiel aux données. Nos algorithmes fonctionnent en temps linéaire par rapport à la taille de l'image et le temps d'exécution est constant par rapport à la taille de l'élément structurant (SE). La combinaison de ces deux propriétés peut être héritée aux opérateurs composés par leur concaténation.

Nous allons également démontrer comment adapter nos algorithmes à des cas spécifiques tels que les filtres spatialement variants ou le calcul des éléments structurants séparables.

Dans l'ensemble, nos algorithmes nécessitent peu de mémoire, ce qui facilite leur implémentation sur des systèmes avec des contraintes d'espace telles que des appareils embarqués ou mobiles, des caméras intelligentes, etc. Dans la partie « Implémentations », nous allons montrer que les performances de nos algorithmes peuvent largement dépasser celle des librairies de références comme OpenCV [146]. Cela ouvre de nombreuses perspectives d'intégration de nos algorithmes dans des applications critiques en latence de calcul : vision pour les voitures autonomes ou pour l'équipement portable militaire.

Quelques considérations pratiques

Dans la pratique, la construction des opérateurs morphologiques repose sur des concaténations longues des filtres atomiques de l'érosion et de la dilatation [157]. Ces concaténations longues sont caractérisées par les variations des paramètres d'un étage à l'autre. Cela pose des sérieux problèmes aux implantations efficaces pour deux raisons :

- Elles augmentent fortement les besoins en mémorisation. Souvent, chaque résultat intermédiaire doit être enregistré.

- Elles se caractérisent par une importante latence de calcul. Ce critère est souvent oublié dans les évaluations des algorithmes, mais il peut devenir critique. Dans le contexte des paramètres des opérateurs morphologiques variables, l'optimisation de la latence peut devenir très complexe et devenir même un verrou technologique.

Si de nombreux algorithmes plutôt rapides ont été proposés. À chaque fois, ces algorithmes se concentrent essentiellement sur la réduction de la complexité de calcul, sans toutefois s'intéresser au problème de la latence. Souvent même pour réduire le nombre de comparaisons, les algorithmes, dits rapides, augmentent la latence en augmentant les besoins de mémorisation intermédiaire [99].

En collaboration avec le Centre de Morphologie Mathématique des Mines-ParisTech, nous avons travaillé sur un cadre algorithmique permettant de proposer des opérateurs atomiques à complexité constante, engendrant une faible latence et minimisant des besoins de mémorisation. Le traitement à flot de données sans mémorisation intermédiaire est la ligne conductrice de notre démarche.

Algorithmes réputés efficaces

Nous avons déjà mentionné que les algorithmes optimisés visent, en général, à réduire la redondance des calculs en utilisant des structures de données adaptées pour conserver les résultats intermédiaires.

Pour comparer les algorithmes, nous entendons par la *la latence* la distance temporelle entre les mêmes positions dans les deux flux. Considérons un système $Y = f(X)$ avec X et Y les flux de données d'entrée et de sortie. Ainsi la latence devient une valeur sans dimension, exprimée en nombre d'échantillons de données. C'est la somme de plusieurs facteurs :

— ***latence d'opérateur*** : le retard induit par des opérateurs non causaux du fait que la valeur à la sortie dépend des échantillons de signaux futurs. Par exemple, un filtre max de base $y_j = max(x_{j-w/2}, \ldots, x_{j+w/2})$. On ne peut pas afficher y_j avant d'avoir lu tous les x_i jusqu'à $X_{j+w/2}$, avec w la largeur SE (ou zone en 2-D). Il représente le temps de réponse du système.

— ***latence d'algorithme*** : c'est un retard, produit par l'algorithme, nécessaire pour produire le résultat après que toutes les données d'entrée nécessaires sont disponibles. Il est étroitement lié à la mémorisation des résultats intermédiaires. Il est évident qu'une grande latence résulte d'un stockage important.

Une des approches explorées est celle basée sur l'histogramme. Il s'agit de l'approche initialement utilisée par Huang *et al.* [114] pour le filtrage médian, puis par Chaudhuri *et al.* [84] pour le filtrage des rangs et plus tard par Van Droogenbroeck et Talbot [100] pour les opérateurs morphologiques à forme arbitraire. Les auteurs utilisent un histogramme pour représenter les valeurs contenues dans le SE. Pendant la translation du SE sur l'image, l'histogramme est mis à jour par inclusion / suppression des valeurs d'entrée / de sortie. Si la latence de l'algorithme peut être évaluée comme nulle, l'efficacité de l'utilisation de l'histogramme est liée au codage des pixels, définissant le nombre de niveaux de couleurs, et elle devient très problématique pour les nombres à virgule flottante.

L'approche par décomposition de SE offre une autre possibilité d'obtenir une mise en œuvre rapide des SE complexes non seulement sur le matériel spécialisé, mais aussi sur les ordinateurs séquentiels. L'optimisation est obtenue en divisant l'effort de calcul grâce à deux aspects-clefs indépendants : i) une séparation des dimensions ou des tailles du SE et ii) l'algorithme rapide utilisé pour calculer les opérations à dimension réduite.

La décomposition la plus simple est la *décomposition linéaire*. Elle provient de l'associativité de la dilatation [131]. Pecht [148] a proposé une décomposition logarithmique efficace basée sur le *l'ensemble extrême* de certains SE. Van den Boomgaard et Wester [73] montrent que la décomposition de Pecht peut être améliorée pour les formes convexes. Ils proposent une décomposition d'une forme arbitraire par l'union des formes convexes issues d'une collection fixe, des formes décomposables efficacement. Coltuc et Pitas [89] proposent un algorithme basé sur la factorisation efficace pour les tailles des SE 2^n.

Dans la diversité des approches, ce sont les décompositions en une dimension (1-D) qui s'avèrent les plus explorés.

Algorithmes 1-D

L'un des algorithmes 1-D les plus anciens et les plus utilisés est l'algorithme de van Herk [111] proposé en 1992. Le même algorithme complété par des preuves théoriques a également été publié par Gil et Werman [106], et plus tard amélioré par Gevorkian *et al.* [103] et Gil et Kimmel [107]. Un autre algorithme similaire a été proposé dans [140]. l'algorithme demande deux passes sur les données d'entrée : causale et anti-causale. Par conséquent, les calculs en flot de données sont impossibles et la latence d'algorithme est de N.

Clienti *et al.* [88] propose une modification intéressante de l'algorithme de Van Herk réduisant la mémoire à $2W$. Il identifie et il propage les extrema sur l'ensemble du SE. Mais toujours deux passes sont nécessaires : causale et anti-causale, par conséquent, la latence de l'algorithme reste N.

Van Droogenbroeck et Buckley [99] publient un algorithme pour l'érosion et l'ouverture 1-D basé sur la détection des ancres (anchors). Cet algorithme offre une accélération très intéressante. Cependant, il faut retenir que pour son implémentation les auteurs utilisent l'histogramme pour détecter le minimum, donc l'implantation présentée encore est une fois limitée aux nombres entiers.

En [125] Lemire propose un algorithme rapide pour un SE 1-D, asymétrique de gauche et plus tard dans [124] pour le SE symétrique. Il peut s'appliquer aux données en virgule flottante et nécessite peu de mémoire et sa latence est nulle. Cependant, le moyen de du stockage intermédiaire des coordonnées locales représente effectivement des accès aléatoires aux données d'entrée. La première proposition de l'algorithme concentre uniquement sur des SE asymétriques, en deuxième version il s'approche de notre démarche.

8.1 Définitions et notions de base

Considérons l'opération de dilatation et d'érosion $\delta, \varepsilon : \mathscr{L} \to \mathscr{L}$ calculées sur la fonction $f \in \mathscr{L}$, avec $f : D \to V$. Supposons $D = \text{supp}(f) = \mathbb{Z}^n$, $n = 1, 2, \ldots$ et $V = \mathbb{Z}$ et $\mathbb{R}$. δ_B, ε_B sont paramétrés par un élément structurant B, supposé être rectangulaire ou plat $B \subset D$, et invariant sous translation.

L'érosion et la dilatation sur des fonctions sont définies par extension de l'addition de Minkowski sur les ensembles.

$$[\delta_B(f)](x) = [\bigvee_{b \in B} f_b](x) \tag{8.1}$$

$$[\varepsilon_B(f)](x) = [\bigwedge_{b \in \widehat{B}} f_b](x) \tag{8.2}$$

où $\widehat{\ }$ représente la transposition de l'élément structurant, égale à la réflexion de l'ensemble $\widehat{B} = \{x \mid -x \in B\}$, et f_b représente la translation de la fonction f par un vecteur $b \in D$.

Les équations Eqs. (8.1, 8.2) peuvent être implémentées comme suit :

$$[\delta_B(f)](x) = \max_{b \in B} f(x - b) \tag{8.3}$$

$$[\varepsilon_B(f)](x) = \min_{b \in B} f(x + b) \tag{8.4}$$

8.2 Algorithme de dilatation 1-D à complexité constante, indépendante de la taille du SE

Dans cette section nous présentons notre contribution au cadre algorithmique pour des opérateurs morphologiques atomiques. Il s'agit d'un algorithme 1-D optimisé pour la dilata-

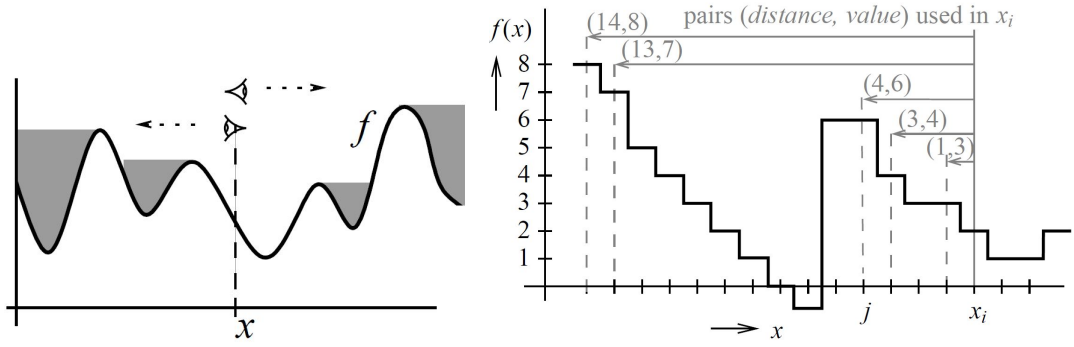
tion et l'érosion, ayant les propriétés suivantes :

- latence de l'algorithme minimale ;
- utilisation de mémoire strictement minimale ;
- adapté aux éléments structurants centrés ou asymétriques ;
- à complexité constante par rapport à la taille d'élément structurant ;
- à complexité linéaire par rapport à la taille de l'image ;
- autorisant les calculs à la volée et respectant le sens de balayage de l'image ;
- choisir librement la position du centre de l'élément structurant.

Le principe de l'algorithme présenté est basé sur un codage efficace du profil de la fonction d'entrée permettant de réduire le nombre de comparaisons et traiter des données à la volée. Le grand avantage de cet algorithme est la possibilité de paramétrer la position du centre du SE pour des SE asymétriques ou centrés.

8.2.1 Élimination des valeurs inutiles

Le principe de l'algorithme repose sur l'élimination à la volée de toutes les valeurs qui ne prendront jamais part dans le résultat du Max ou du Min. Car pour calculer $\delta_B f(x)$ nous avons seulement besoin de ces valeurs de $f(x_i)$ que l'on peut voir de x en regardant sur le profil topographique de f. Ainsi, les vallées ombragées par des montagnes contiennent des valeurs inutiles (Fig. 8.1). Remarquons que les valeurs masquées dépendent de f et pas de B [6].



(a) Relief topographique de f (b) Encodage du signal utilisé par l'agorithme.

FIGURE 8.1 – Illustration de la dilatation $\delta_B f(x)$: les valeurs appartenant aux vallées sont cachées par les sommets en regardant le relief topographique de f à partir de la position du x.

Proposition 1. [*Valeurs inutiles - cas causal*] *En cas de dilatation $\delta_B f$ avec $f : \mathbb{Z}^+ \to \mathbb{R}$, par un élément structurant B causal et invariant par translation* [1] *contenant son origine, aucune valeur $f(i)$ telle que $f(i) \leq f(j)$ et $i < j$ n'influence pas la valeur de la dilatation.*

$$\delta_B f(k), \qquad \forall k \geq j \tag{8.5}$$

Preuve. *De l'Eq. 8.3, tout x tel que $i < j \leq x$, si $i \in B(x)$ alors $j \in B(x)$. Si $f(i) < f(j)$, alors $f(i) < \max_{b \in B} f(x-b)$, et $f(i)$ n'a aucun impact sur le résultat de dilatation.*

Cela signifie que toutes les valeurs de $f(i)$ telles que :

$$f(i) \leq f(j), \text{ avec } i < j, \tag{8.6}$$

peuvent être éliminées - oubliées. Une seule comparaison $f(i) \leq f(j)$, réalisée pendant la lecture de $f(j)$, évite les calculs des comparaisons inutiles pour tous les $B(x)$ qui couvrent i et j.

1. p. ex. $B(x+k) = B(x)+k$

Notons deux observations importantes :

1. $\forall k \geq j$ en Eq. 8.5 que toutes les valeurs inutiles j restent inutiles.
2. Utilisant un $B \subset \mathbb{Z}$, borné et causal p. ex. un intervalle $B(x) = [x-b, x]$, avec $b < \infty$, signifie que pour le calcul de $\delta_B(x)$, nous pouvons supprimer les valeurs en dehors de la portée, p. ex. $f(x_i)$, avec $x_i < x - b$.

Transformation parcours anti-causal - causal

Considérons tout SE B, $B \subset D$ équipé par une origine $x \in D$. En assumant un accès séquentiel aux données en entrée, la dilatation $\delta_B f(x)$ dépend non seulement des points lus auparavant, mais aussi de ceux après x. Nous disons que B n'est pas causal.

On peut transformer un SE anti-causal en un SE causal en utilisant la propriété que la dilatation commute avec la translation ($t \in D$)

$$\delta_{B+t} f(x) = \delta_B f(x-t) \tag{8.7}$$

Dans la pratique, cela ce traduit uniquement par une translation de l'écriture du résultat à l'endroit (position) correct.

Codage d'une fonction

De la même façon que des objets binaires qui peuvent être codés en utilisant la distance à leurs frontières, les fonctions peuvent être codées en calculant la distance à chaque changement de la valeur. Observons la Fig. 8.1(b), les flèches indiquent ces valeurs qui entrent dans le calcul de $\delta_B f(x_i)$. Les valeurs non indiquées par une flèche sont plus petites ou égales à $f(j) = 6$ et n'ont aucun impact sur le résultat $\delta_B f(x)$, pour $x \geq j$.

8.2.2 Algorithme de dilatation 1-D

Les échantillons $f(x)$ utilisés dans le calcul $\delta_B f(x_i)$ sont codés par paires (*la distance, la valeur*) comme illustré dans l'image Fig. 8.1(b). Ceci est possible grâce au codage efficace de la fonction d'entrée permettant l'indexation relative des échantillons.

Algorithm 2: Dilatation 1-D à flot de données stricte [6]

```
Input: rp, wp - reading/writing position; F - input signal value (read at rp); SE1, SE2 - SE size
       towards left and right; N - length of the signal; fifo - the FIFO
Result: dF - output signal value (to be written at wp)

// Dequeue all queued smaller or equal values
1 while fifo.back()[1] ≤ F do
2 |   fifo.dequeue()

// Delete too old value
3 if (wp - fifo.front()[2] > SE1) then
4 |   fifo.pop()

// Enqueue the current sample
5 fifo.push({F, rp})

6 if rp = min (N, wp + SE2) then
7 |   return ( fifo.front()[1] ) ;                    // return a valid value
8 else
9 |   return ({}) ;                                   // return empty
```

Latence

La latence globale de l'algorithme est une fonction de deux facteurs : i) la latence de la dilatation 1-D et ii) la latence d'opérateur.

Dilatation 1-D : elle commence à produire le résultat dès que la position de lecture rp atteint la dernière position couverte par l'élément structurant. Ceci correspond à la latence d'opérateur. Par conséquent la latence de l'algorithme est nulle.

Complexité de calcul

L'algorithme (Fig. 4) contient une séquence des opérations $\mathcal{O}(1)$ et une la boucle *while* des lignes 1-2) pour suppression des valeurs inutiles. C'est une partie dépendante des données. Cette boucle *while* est exécutée en moyenne une fois par pixel, faisant la complexité moyenne de l'algorithme constant par pixel.

Remarquons que chaque pixel entrant est stocké une seule fois (la ligne 5). Chaque pixel est supprimé une seule fois pour toute : i) quand il devient "trop ancien" (les lignes 3-4) ou ii) quand il est masqué par un autre point plus grand (les lignes 1-2).

Utilisation de mémoire

L'algorithme d'érosion 1-D ne stocke que les données trouvées dans le périmètre de l'élément structurant (un concept similaire est également utilisé dans [99]), c'est-à-dire pour calculer $\varepsilon_B f(x)$, seulement $F(x_i)$, avec $x_i = B(x)$ sont nécessaires. De même, pour la dilatation, il suffit de stocker les données couvertes par transposé $\widehat{B}(x)$. En 1-D, la taille de la FIFO est donc délimitée par le cardinal de B.

La comparaison de propriétés des l'Alg. 4 proposé avec l'état de l'art est résumée dans le Tableau 8.1.

TABLEAU 8.1 – Comparaison des propriétés des algorithmes 1-D rapides publié dans [6].

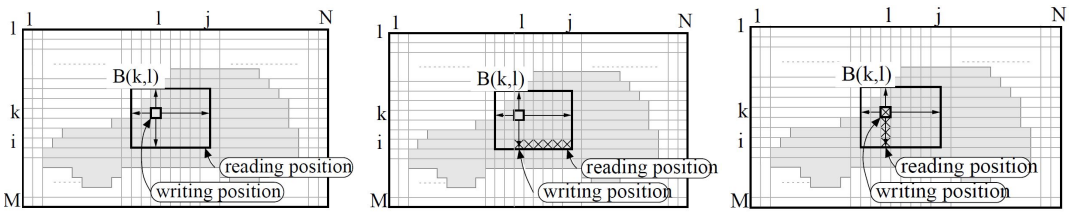
Algorithme	SE type	Comparaisons par pixel	Latence Algorithme	Mémoire
Naive 1-D	User	$W-1$	0	N
van Herk Gil-Wermann	Sym	$3-\frac{4}{W}$	W	$N+2W$
Gil-Kimmel	Sym	$1,5+\frac{\log_2 W}{W}+\mathcal{O}(\frac{1}{W})$	W	$N+3W$
Lemire	Left	3	0	$N+W$
Lemonnier	Sym	nc	N	$2N$
Van Droogenbroeck-Buckle	Sym	nc	0	$2N+G$
Algorithme proposé	User	$\mathcal{O}(1)$	0	$2W$

Sym = symétrique SE ; Left = anticausal SE ; User = forme libre ; W=taille SE ; N=taille de ligne ; G = nombre de niveaux de gris ; nc = non communiqué.

8.2.3 De la 1-D à la 2-D

Selon le principe de décomposition de l'élément structurant, le calcul du SE final en 2-D est réalisé "en itérant" des calculs en dimension plus petite (1-D). Pour la plupart des algorithmes, cela implique le stockage de données intermédiaire, souvent de l'image entière.

L'algorithme proposé (Alg. 4) peut exécuter les calculs dans les différentes dimensions en même temps, à la volée, et par conséquent éliminer la nécessité de stockage de données intermédiaires et réduire la latence globale.



(a) SE rectangle (b) Calcul de la partie horizontale (c) Calcul de la partie verticale

FIGURE 8.2 – Séparation du calcul, $\times\times$ représente les données intermédiaires stockées.

Ainsi, l'image d'entrée est lue dans l'ordre d'acquisition, ligne par ligne. Chaque ligne est dilatée horizontalement. Le résultat de la dilatation horizontale est immédiatement injecté, pixel par pixel, dans la dilatation verticale dans la colonne correspondante (Fig. 8.2).

Algorithm 3: Dilatation 2-D à zéro latence [6]

```
Input: in_stream - input image pixels stream
       M,N - height, width of the image
       SE1, SE2, SE3, SE4 - struct. element size
Result: out_stream
1  const. PAD ← 0 ;                                   // Set the Padding Constant
2  vfifo ← array [1..N] of FIFO ;                     // array of N empty FIFOs for the vertical dilation part
3  line_rd ← 1 ;                                      // read line counter
4  line_wr ← 1 ;                                      // written line counter
   // iterate over all image lines
5  while line_wr ≤ M do
6      hfifo ← FIFO ;                                 // FIFO for the horizontal part
7      col_rd ← 0 ;                                   // read column counter
8      col_wr ← 1 ;                                   // written column counter
       // iterate over all columns
9      while col_wr ≤ N do
           // horizontal dilation on the line_wr line
10         if line_rd ≤ M then
11             if col_rd < N then
12                 F ← in_stream.read()
13             else
14                 F ← PAD ;                          // Padding constant
15             col_rd ← min(col_rd +1, N)
16             dFx ← 1D_Dilation (col_rd, col_wr, F, SE1, SE3, N, hfifo)
17         else
18             dFx ← PAD ;                            // Padding constant
           // vertical dilation of the col_wr column
19         if dFx ≠ {} then
20             dFy ← 1D_Dilation (min(line_rd,M), line_wr, dFx, SE2, SE4, M, vfifo[col_wr ])
21             if dFy ≠ {} then
22                 out_stream.write(dFy)
23             col_wr ← col_wr +1
24     line_rd ← line_rd+1
25     if dFy ≠ {} then
26         line_wr ← line_wr +1
```

Latence

La latence globale de l'algorithme est fonction de deux facteurs : i) latence de la dilatation 1-D et ii) latence de la décomposition 2D. Dilatation 2D : la latence d'algorithme de la décomposition 2D est nulle si les données sont gérées dans le flot.

TABLEAU 8.2 – Comparaison des algorithmes 2-D rapides, publié dans [6]

Algorithm	Type de SE	Latence	Mémoire de données	Mémoire intermédiaire	Complexité per pixel
Naive 2-D	User	0	MN	0	$\mathcal{O}(WH)$
Urbach-Wilkinson	User	MN	$2MN$	$NH\log_2 W$	$\mathcal{O}(N_c + \log_2(L_{max}(C)))$
Van Droogen-broeck-Talbot	User	0	NH	WHG	$\mathcal{O}(H\log_2(G))$ * * square SE
Algorithme 3	Rect.	0	0	$2(NH+W)$	$\mathcal{O}(1)$

$W \times H$ = taille SE (W-largeur×H-hauteur) ; $N \times M$ = taille de l'image ; G = nombre de niveau de gris ; $L_{max}(C)$ = longueur maximale de "chord" ; N_c = nombre des chords.

Complexité de calcul

La complexité de l'algorithme 2D est $\mathcal{O}(N)$, où N est le nombre de pixels dans l'image. À chaque coordonnée, le 2D_Dilation appelle deux fois l'algorithme 1-D (Fig. 4) : une fois pour la dilatation verticale et une fois pour la partie de dilatation horizontale.

Utilisation de la mémoire

En 2-D, les exigences de mémoire sont données par la décomposition du rectangle comme $R = H \oplus V$ (R = rectangle, H / V = segment horizontal / vertical respectivement) et par le fait que le rectangle glisse sur l'image en mode balayage. Cela signifie que la partie verticale de la dilatation 2-D s'exécute dans toutes les colonnes simultanément, un pixel par colonne à la fois. Graphiquement, les exigences en mémoire correspondent au stockage des données d'image des lignes actuellement intersectées par le SE.

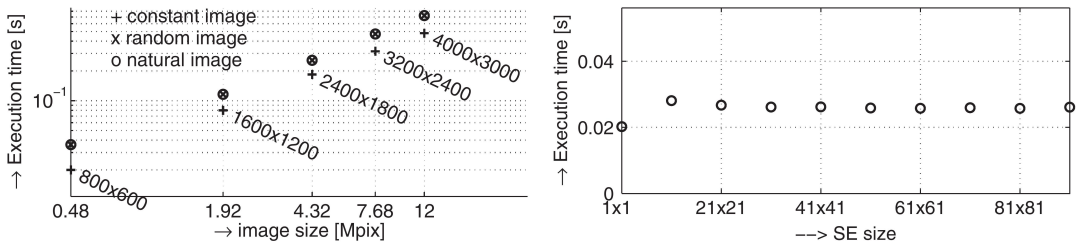
Considérons une érosion (ou une dilatation) d'une largeur d'image $N \times M$ par $W \times H$ rectangulaire (largeur par hauteur) SE. Les exigences de mémoire sont les suivantes :

- pour la dilatation horizontale, on a une mémoire de taille $2W$
- et N fois pour la dilatation dans le sens vertical, de taille $2H$

8.2.4 Évaluation

La Fig. 8.3(a), illustre le temps d'exécution, mesuré pour les images à contenus différents (sur un processeur Intel Core 2 2GHz, avec 2 Go 800MHz RAM double port, sous Linux). Pour le test, nous avons utilisé trois images : une image aux valeurs constantes (le meilleur cas), une image remplie de bruit blanc et une image naturelle pour analyser la retombée de la partie d'algorithme 1D dépendante des données. Dans tous les cas, le temps d'exécution augmente de manière linéaire par rapport à la taille de l'image d'entrée. La Fig. 8.3(b) illustre bien que le temps d'exécution est constant et indépendant de la taille de l'élément structurant.

La Fig. 8.4 compare les performances avec des algorithmes 2-D choisis. Les mesures obtenues confirment la complexité linéaire de l'algorithme. De plus, les temps de traitement obtenus sont comparables à celles des algorithmes réputés les plus efficaces.



(a) Temps d'exécution par rapport à la taille de l'image

(b) Temps d'exécution par rapport à la taille de l'élément structurant

FIGURE 8.3 – Évaluation des propriétés de l'algorithme 2-D par décomposition

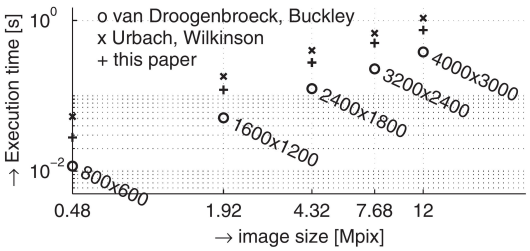


FIGURE 8.4 – Comparaison des performances, élément structurant de taille 21×21, taille de l'image 800x600

8.3 Extensions

Les extensions présentées ont été étudiées dans les travaux de thèse de Jan Bartovsky [69] que j'ai co-encadré.

8.3.1 Opérateurs morphologiques sous angle arbitraire

L'algorithme 1-D proposé peut être directement utilisé pour obtenir des éléments structurants 1-D inclinés sous des angles arbitraires. Pour cela, les pixels lus dans le flot sont à la volée mappés dans des couloirs (Fig. 8.5) selon l'algorithme de Bresenham [77]. La lecture des pixels dans l'ordre reste préservée. Les seuls éléments utilisés de mémoire sont les files d'attente dont la profondeur et le nombre dépendent uniquement de la longueur du SE et de la taille (largeur) de l'image [21].

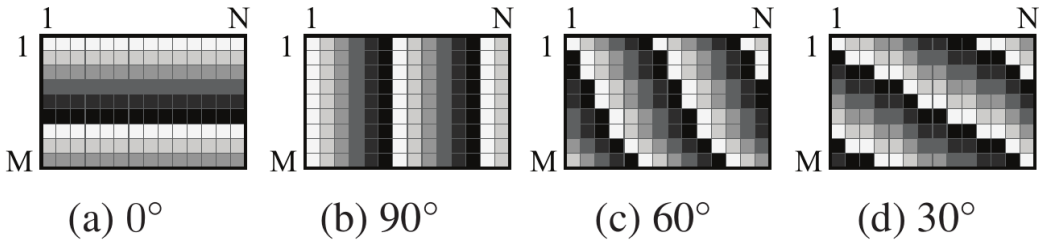


FIGURE 8.5 – Exemple des couloirs obtenus par l'algorithme de Bresenham pour différents angles.

8.3.2 Concaténations profondes à latence minimale

Dans la pratique, les dilatations et les érosions se combinent pour former d'autres opérateurs. Nous présentons dans la suite les bénéfices de notre contribution algorithmique pour des concaténations optimales et avec des performances obtenues inégalées pour certaines.

Filtres Alternés Séquentiels

Les Filtres Alternés Séquentiels (ASF[2]) représentent un exemple d'opérateurs à concaténation profonde par excellence. La mise en œuvre d'un ASF^{λ} est donnée par Eq. 8.8 :

$$ASF^{\lambda} = \delta_{B_i}\varepsilon_{Bi\oplus Bi}\delta_{B_i}\ldots\delta_{B_1}\varepsilon_{B1\oplus B1}\delta_{B_1} \tag{8.8}$$

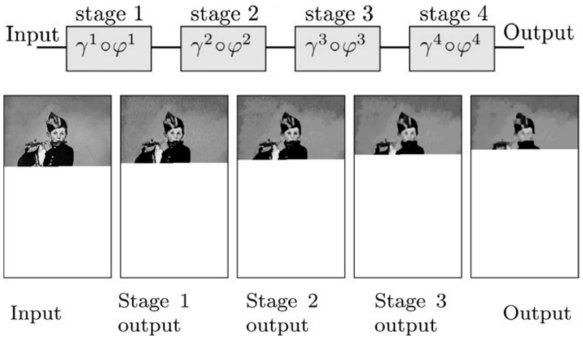


FIGURE 8.6 – Calcul d'un filtre ASF sans mémorisation intermédiaire.

Un tel ASF peut être maintenant être calculé à la volée dans le sens du balayage normal de l'image d'entrée. La position d'écriture de l'opérateur précédent dans la cascade devient la position de lecture de l'opérateur suivant (Fig. 8.6). Tous les opérateurs peuvent s'exécutent simultanément. Il n'y a pas de stockage intermédiaire de données entre les étapes car les résultats intermédiaires sont pipelinés [6] et la latence de calcul est réduite au minimum.

Approximation des polygones

La séparabilité de la dilatation morphologique n-D en dimensions inférieures est une propriété bien connue et déjà mentionnée dans ce document. Les dilatations par polygones convexes réguliers P_{2n} peuvent être décomposées en séquence des dilatations linéaires 1-D selon l'équation suivante :

$$\delta_{P_{2n}}(f) = \underbrace{\delta_{L_{\alpha_1}}(\ldots\delta_{L_{\alpha_n}}(f))}_{\text{n-times}} \tag{8.9}$$

où $\alpha_i = (i-1)\frac{180^\circ}{n}[^\circ]; i \in \mathbb{N}, i \leq n$ et $\|L_{\alpha_i}\| = 2Rsin\left(\frac{180^\circ}{2n}\right)$, R étant le cercle englobant du polygone.

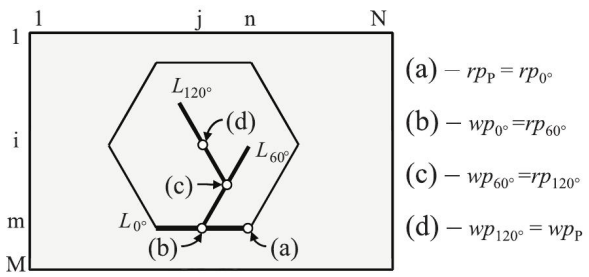


FIGURE 8.7 – Décomposition d'élément structurant hexagonal en trois éléments structurants linéaires inclinés. rp_α représente la position de lecture de la dilation 1-D à angle α ;wp_α représente la position d'écriture dy résultat de la dilation 1-D à angle α.

2. ASF - Alternate Sequential Filter

L'implémentation basée sur l'approche algorithmique proposée conserve les mêmes propriétés, telles que la complexité de calcul linéaire par rapport à la taille de l'image et indépendante de la taille du SE. La latence est toujours strictement égale à la latence de l'opérateur, c'est-à-dire qu'il est dépendant de la taille de l'élément structurant utilisé. On utilise toujours un accès strictement séquentiel aux données. La consommation de mémoire est très inférieure à la taille de l'image [3].

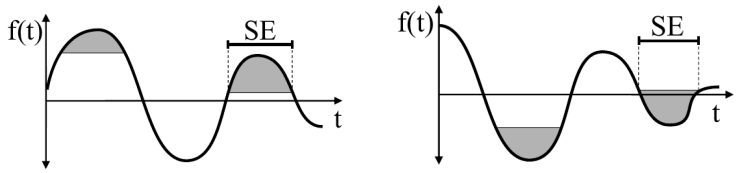
8.3.3 Ouverture morphologique 1-D à complexité constante en flot de données

Cette contribution est issue des travaux menés avec notre doctorant, Jan Bartovsky et Mines-ParisTech. Dans [21], nous introduisons un nouvel algorithme pour l'opérateur de l'ouverture morphologique 1-D. Il se distingue par une complexité linéaire par rapport à la taille d'élément structurant, par les calculs en flot de données et par la possibilité de fournir les résultats de granulométries en parallèle [132]. Nous pouvons citer quelques autres algorithmes efficaces pour le calcul direct de l'ouverture : notamment Soille al. [165], Van Droogenbroeck et Buckley [99], Urbach et Wilkinson [171] et Morard [141]. Cependant, on peut démontrer leurs faiblesses dans le cas des SE orientés sous un angle arbitraire.

Nous considérons les opérateurs de l'ouverture et de la fermeture morphologiques, appliqués aux images à niveaux de gris $\varphi_B, \gamma_B : \mathbb{Z}^2 \to \mathbb{R}$, définis comme suit :

$$\varphi_B(f) = \varepsilon_B[\delta_B(f)]; \gamma_B(f) = \delta_B[\varepsilon_B(f)] \tag{8.10}$$

L'ouverture coupe littéralement les "sommets" plus étroits que le SE . La fermeture remplit les "vallées" plus étroites que SE (Fig. 8.8).



(a) Ouverture 1-D (b) Fermeture 1-D

FIGURE 8.8 – Illustration du principe d'ouverture et de fermeture

Nous proposons un algorithme basé le découpage du sommet récursivement, de haut en bas. Le signal d'entrée est continuellement analysé, dans l'ordre de balayage de l'image afin d'identifier l'existence d'un sommet en fonction des différentes configurations (Fig. 8.9). La valeur du sommet rasé $f(x)$ est remplacée par $f(x-1)$ dans la configuration (a), ou par $f(x+1)$ dans la configuration (b), respectivement. Après la suppression du sommet en cours, tout le processus sera répété si, et seulement si, il s'agit de la configuration (c).

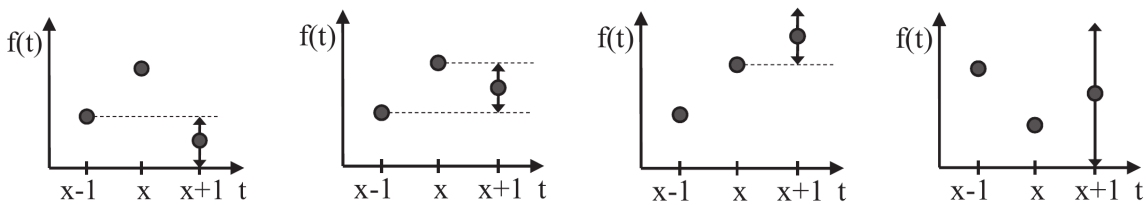


FIGURE 8.9 – Quatre configurations utilisées pour détecter les "sommets" du signal.

Pour assurer le bon fonctionnement de la méthode, on propose un autre encodage particulier du signal d'entrée. Le principe est représenté sr la Fig. 8.10, où le signal d'entrée est codé par paires {valeur, position du dernier pixel sur le plateau}. Cet encodage et qu'il peut être construit à la volée et qu'il permet également de supprimer un plateau-sommet en une seule opération.

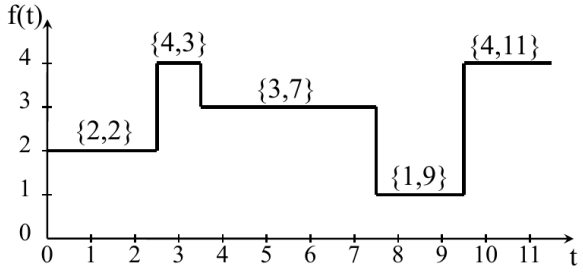
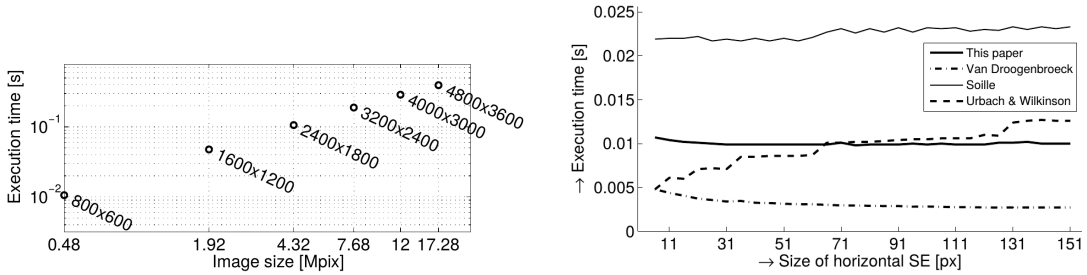


FIGURE 8.10 – Encodage du signal pour l'algorithme récursif d'élimination des sommets.

Nous avons évalué le temps d'exécution par rapport à la taille du SE horizontal et par rapport à la taille de l'image. Sur la figure 8.11, nous pouvons constater que l'algorithme d'ouverture 1-D respecte les propriétés du cadre algorithmique présenté. Il est indépendant de la taille de l'élément structurant et il dépend linéairement de la taille de l'image. Les benchmarks ont été réalisés sur un processeur Intel Xeon E5620 @ 2.4GHz sous Linux.



(a) Temps d'exécution par rapport à la taille de l'image

(b) Temps d'exécution par rapport à la taille de l'élément structurant

FIGURE 8.11 – Évaluation des propriétés de l'algorithme de l'ouverture 1-D.

Comparé avec d'autres algorithmes (Fig. 8.12), les algorithmes d'ouverture de Morard et Van Droogenbroeck surpassent les performances de notre solution. En revanche, nous pouvons garantir les mêmes performances pour des ouvertures sous angles arbitraires et le calcul simultané des granulométries.

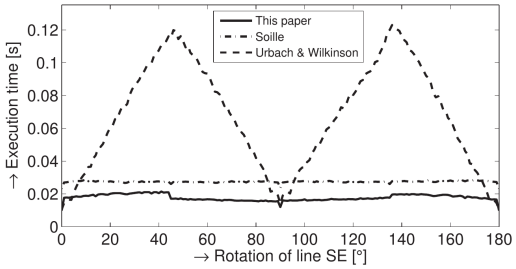


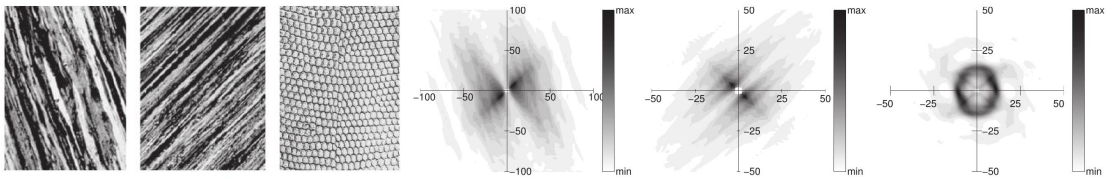
FIGURE 8.12 – Évaluation des performances des algorithmes d'ouverture linéaire à angle arbitraire.

Calcul simultané des granulométries

Supposons les éléments structurants linéaires inclinés sous forme B_L^α où L indique la taille (longueur) du SE et α l'angle de rotation. Nous appelons la distribution des tailles $PS(\alpha, L)$ l'ensemble des granulométries paramétrées par la taille du SE et la rotation du SE. La distribution des tailles est par la suite définie comme suit :

$$[PS(\alpha, L)](f) = \Sigma_D \left(\gamma_{B_L^\alpha} f - \gamma_{B_{L+1}^\alpha} f \right) \tag{8.11}$$

Grâce au fait que l'algorithme d'ouverture élimine un sommet successivement niveau par niveau, il est très facile d'obtenir le nombre de pixels du même niveau et de les accumuler à la volée [21].



(a) Texture 1 (b) Texture 2 (c) Texture 3 (d) *PS* of texture 1 (e) *PS* of texture 2 (f) *PS* of texture 3

FIGURE 8.13 – Analyse des textures à l'aide de la distribution des tailles [52].

Nous obtenons ainsi le spectre PS avec un effort supplémentaire minimal. Le principe avec l'algorithme détaillé est très bien illustré dans le manuscrit de thèse de Jan Bartovsky [69].

Ceci permet d'atteindre des performances inégalées pour des applications pratiques nécessitant d'obtenir des mesures exhaustives. Nous pouvons illustrer le gain sur des exemples d'analyse d'orientation [52]. Nous faisons varier un des deux paramètres α, L et nous mesurons les granulométries. Par exemple, pour l'application sur la Figure 8.13, nous estimons $\Sigma_{\forall L} PS$ par rapport à α.

Notre algorithme qui calcule le spectre de la pente dans un balayage d'image unique, il permet d'introduire une accélération significative par rapport à l'approche par résidu nécessitant des ouvertures multiples.

Algorithm 4: Ouverture 1-D à complexité constante avec le calcul simultané de la distribution de tailles [52].

Input: F - input sample $f(rp)$; rp - current reading position; L - SE size; Q - Queue
Result: Y - sample of $y(rp-L)$; $S[]$ - Size spectrum
Data: Q - a Double-End Queue
Q.back(1)[] - accesses the last enqueued pair $\{F, rp\}$
Q.back(2)[] - accesses the second to the last enqueued pair $\{F, rp\}$

```
1  while F ≤ Q.back(1)[1] do
2    if F = Q.back(1)[1] then
3      Q.dequeue() ;                                   // Remove equal values
4      break ;
5    else
6      if Q.back(2)[1]<Q.back(1)[1] then
7        if F < Q.back(2)[1] then
8          S[Q.back(1)[2] - Q.back(2)[2]] += Q.back(1)[1] - Q.back(2)[1] ;
           // Accumulate discarded peak in size spectrum array S
9          Q.back(2)[2] = Q.back(1)[2] ;                // Config. (a)
10       else
11         S[Q.back(1)[2] - Q.back(2)[2]] += Q.back(1)[1] - F ; // Accumulate peak
           in S
12       Q.dequeue() ;                                 // Discard peak, conf. (b), (a)
13     else
14       break ;                                       // Configuration (d)
15 Q.push({F, rp}) ;                                   // Enqueue current sample
16 if rp = Q.front()[2] + L then
17   Q.pop() ;                                         // Delete outdated value
18 if rp ≥ L then
19   return (Q.front()[1] ) ;                          // Return opening sample
```

8.4 Opérateurs morphologiques spatialement variants

Les applications à base d'éléments structurants variant spatialement surpassent les éléments invariants par leur capacité à s'adapter localement au contenu de l'image.

Nous nous inspirons du principe appliqué aux dilatations et aux érosions binaires [110]. Nous proposons son extension à la morphologie fonctionnelle : morphologie à variation spatiale 1-D [32] et approximations de rectangles 2-D variants [13].

Dans l'approche proposée, nous retrouverons les propriétés algorithmiques exploitées dans les parties précédentes : latence d'algorithme nulle, optimisation de mémoire grâce à l'accès strictement séquentiel aux données.

Notions importantes

Considérons les opérations de dilatation et d'érosion $\delta, \varepsilon : \mathcal{L} \to \mathcal{L}$ qui sont calculées sur la fonction $f \in \mathcal{L}$ définie comme $f : D \to V$. Supposons $D = \text{supp}(f) = \mathbb{Z}^2$ et $V = \mathbb{R}$ ou $\mathbb{Z}$. δ, ε sont paramétré par un B supposé plat, $B \subset D$. Dans le cas d'opérations spatialement variantes $B : D \to \mathcal{P}(D)$, avec $\mathcal{P}$ représentant un ensemble des sous-ensembles.

Ici, $B(x)$ choisit $\forall x \in D$ une forme $B(x) \in \mathcal{C}$ où $\mathcal{C}$ représente la classe des formes autorisées. Puis la dilatation et l'érosion par un élément structurant plat, spatialement variant s'écrivent :

$$[\delta_B(f)](x) = \bigvee_{b \in B(x)} f(b) \tag{8.12}$$

$$[\varepsilon_B(f)](x) = \bigwedge_{b \in \hat{B}(x)} f(b) \tag{8.13}$$

où $\hat{B}$ signifie la transposition de B. Pour les SE variant spatialement, la transposition s'écrit :

$$\hat{B}(x) = \{y | x \in B(y)\} \tag{8.14}$$

Pour former les opérateurs concaténés tels que l'ouverture et la fermeture, le calcul direct de l'éq. (8.14) nécessite une recherche exhaustive dont la complexité est $\mathcal{O}(N)$, avec $N = Card(D)$. Implémenter l'éq. (8.13) aura ainsi une complexité $\mathcal{O}(LN)$ avec $L =$Card(supp(B)). Par conséquent, implémenter les opérations à base d'adjonctions selon l'éq. (8.12-8.13) est d'une complexité $\mathcal{O}(LN^2)$.

Une autre possibilité pour obtenir les adjonctions se base sur la représentation à l'aide des intersections :

$$[\delta_B(f)](x) = \wedge\{v \in V | \hat{B}(x) + v \geq f\} \tag{8.15}$$

$$[\varepsilon_B(f)](x) = \vee\{v \in V | B(x) + v \leq f\} \tag{8.16}$$

Nous combinons donc des équations (8.12 et 8.16) et nous proposons une approche de calcul à complexité linéaire, indépendante de SE, calculée en flot de données et avec $\mathcal{O}(1)$ et l'utilisation de mémoire minimale.

Restrictions

Les restrictions qui permettent de garantir les bonnes propriétés de l'algorithme sont les suivantes :

— **Continuité** : étant donné que nous avons uniquement la connaissance locale du B et de l'image, le changement de taille du B doit être borné. Autrement, des artefacts peuvent se former.

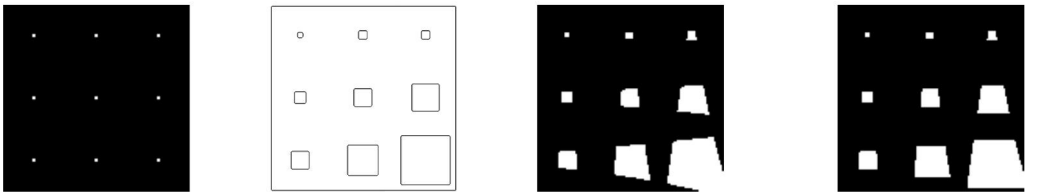
$$\|\frac{\Delta B}{\Delta x}\|_{\mathscr{L}_\infty} \leq 1 \ , \ \|\frac{\Delta B}{\Delta y}\|_{\mathscr{L}_\infty} \leq 1 \text{ and } \frac{\Delta B_{up}}{\Delta x} = 0 \ , \ \frac{\Delta B_{down}}{\Delta x} = 0 \tag{8.17}$$

— **Séparabilité** : les SE variant spatialement ne sont pas facilement séparables en 1-D dimensions. Par exemple, une décomposition de SE rectangulaire spatialement variant ne produit pas de rectangles (Fig. 8.14).

Ces restrictions énoncées limitent l'amplitude de la variation des SE. En 2-D ceci implique que le SE peux varier uniquement en une dimension.

En cas de non-respect des restrictions, nous perdons l'adjonction (et par conséquent l'idempotence), mais nous conservons d'autres propriétés importantes telles que le principe min / max, (anti-) extensibilité et croissance, et en même temps, nous pouvons préserver les détails et nous rapprocher visuellement du résultat exact (Fig. 8.17).

L'algorithme 1-D développé est détaillé en annexe du [32]. Il repose encore une fois sur l'utilisation des FIFO interne contenant des triplets (Fval, rp, dd) - valeur du pixel, position de lecture en cours et distance de dilatation. La gestion de changement de taille du SE est



(a) Original (b) SV SE (c) exact (d) approximation

FIGURE 8.14 – Illustration du problème de séparabilité des éléments structurants spatialement variants. On considère l'opération de la dilatation.

réalisée uniquement par l'adaptation à la volée des positions de lecture et de la position de l'écriture du signal d'entrée.

Les données d'entrée et de sortie sont lues et écrites séquentiellement dans un flux, dans l'ordre d'acquisition. Chaque échantillon est lu et écrit une seule fois. La Fig. 8.15 confirme le temps d'exécution linéaire par rapport au nombre des pixels traités.

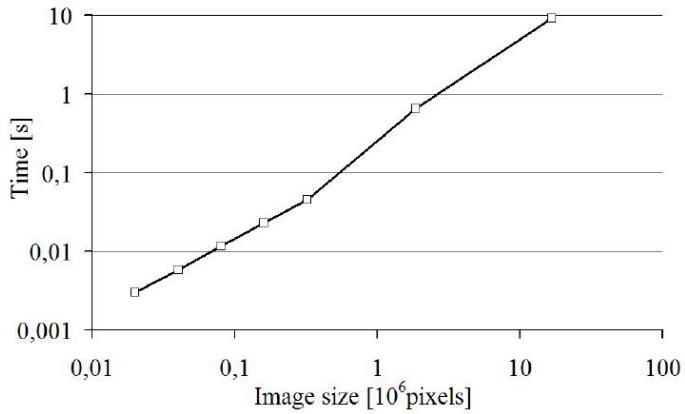


FIGURE 8.15 – Évaluation de performances par rapport à la taille de l'image [32]

L'extension en 2-D se fait par la séparation du calcul de la partie horizontale et verticale, apportant les restrictions mentionnées précédemment. Si les restrictions ne sont pas respectées, notre algorithme va résulter en approximation (d) du résultat exact (c) illustré sur la Fig. 8.14.

Applications

Filtrage des scènes sous perspective

La détection des plaques d'immatriculation est désormais un problème classique de traitement d'images. L'exemple dans la Fig. 8.16 est inspiré de [95], [152]. Sur l'image à droite, on constate une amélioration possible des détections grâce aux éléments structurants adaptés à la perspective de prise vue.

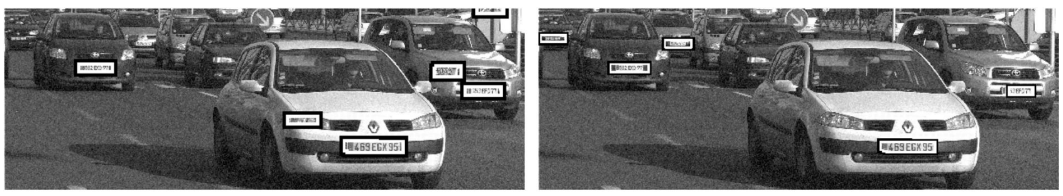
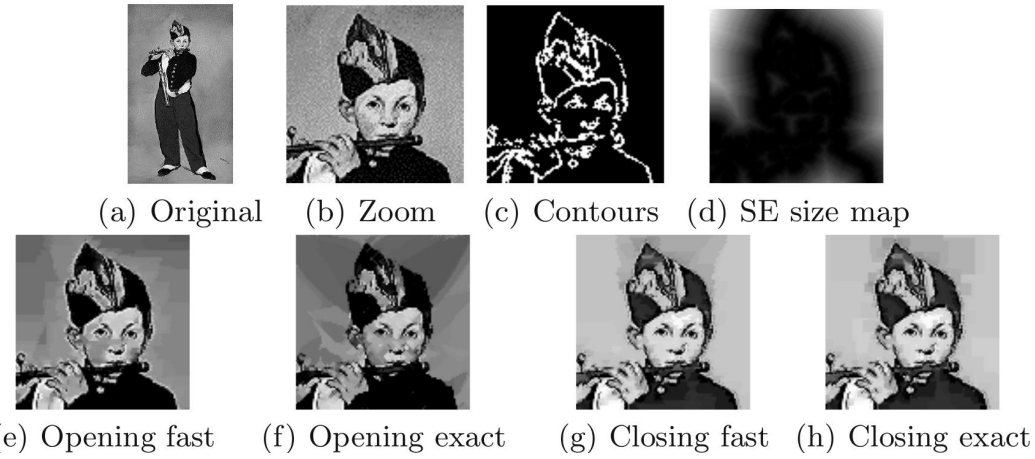


FIGURE 8.16 – Détection des plaques d'immatriculation dans des séquences avec de la perspective.

Filtrage adaptatif

Il peut être utilisé pour le prétraitement d'image sensible au contenu, par ex. une simplification de l'image, réduction du bruit, etc. Pour filtrer efficacement, mais préserver les contours, le filtre a besoin de grands éléments structurants, mais qui ne traversent pas les contours (Fig. 8.17). Les contours peuvent être détectés par n'importe quel opérateur habituel (c). Ensuite, on peut concevoir une carte des tailles des SE rectangulaires avec le côté à taille égale à la moitié de la distance aux contours, donnée par (d). Ainsi nous allons pouvoir préserver les contours et les détails importants. Le reste de l'image est fortement simplifié.



(a) Original (b) Zoom (c) Contours (d) SE size map

(e) Opening fast (f) Opening exact (g) Closing fast (h) Closing exact

FIGURE 8.17 – Illustration des opérateurs de l'ouverture et de la fermeture adaptatives contrôlées par la distance aux contours.

Troisième partie

Implémentations efficaces

Chapitre 9

Méthodes basées sur les équations aux dérivées partielles

Les méthodes de traitement d'images décrites par les équations aux dérivées partielles offrent des avantages d'une modélisation mathématique de la physique sous-jacente des phénomènes décrits, d'une précision sous-pixellique, de l'isotropie et de l'extension directe aux dimensions supérieures.

Leur déploiement dans des applications pratiques a été ralenti par un considérable effort de calcul que demandent les contraintes de stabilité des solutions numériques : complexité des approximations numériques, nombre d'itérations extrêmement élevé et inconnu à l'avance, calculs en virgule flottante.

L'importance de trouver une solution d'implantation efficace peut être perçue de l'effort de recherche centré sur ce sujet. Ceci malgré l'implantation pratique dans le cadre des ensembles de niveaux, introduite par Osher et Sethian [147] (Chapitre 7).

Des solutions d'accélération de calculs ont été recherchées dans les domaines :

i) Mathématiques - J. Weickert [175] propose un schéma numérique basé sur séparation additive des opérations, stable pour les filtres non-linéaires, permettant une intégration numérique rapide. Cette approche a été ultérieurement étendue aux contours par Smereka [163], et par Goldenberg et al. [109],

ii) Algorithmiques - nous pouvons citer pour exemple l'approche de Precioso et Barlaud qui utilisent des splines [151], la limitation de ce type de solutions est qu'elles sont adaptées à un seul type de problème.

iii) Implémentations sur un matériel dédié aux calculs hautes performances - nous pouvons citer les premières approches de Holmgren et Wallin [112] et Sethian [159] font appel aux équipements des calculs de hautes performances (super-ordinateurs). Rumpf et Strzodka[153], Cates et al. [81] ou Sigg et al. [162] ciblent les processeurs graphiques. Hwang et al. [115] proposent une architecture orthogonale généraliste pour solution numérique des EDP [1], Gijbels et al. [105] proposent une architecture matérielle dédiée pour le filtrage d'images à base de la diffusion non-linéaire.

Le problème résiduel de ces approches est qu'elles s'adressent à chaque fois à une application ou opérateur unique. Par exemple, la généralisation n'est pas possible à la fois au bas niveau et à haut niveau de traitement.

Dans [5], nous proposons un cadre unificateur pour la mise en œuvre d'une architecture programmable dédiée aux méthodes des équations à dérivées partielles, basées sur des techniques des ensembles de niveaux. Nous exploitons le fort potentiel de parallélisation de l'algorithme Massive Marching pour proposer une plate-forme multiprocesseur évolutive,

1. Equations aux dérivées partielles

asynchrone, adaptée aux solutions embarquées.

9.1 Classification des motifs de calcul

Notre contribution [5, 53] est double. La première est une classification des techniques des ensembles de niveaux du point de vue de la conception du système (Fig. 9.1). La deuxième est une proposition de l'architecture matérielle programmable et évolutive.

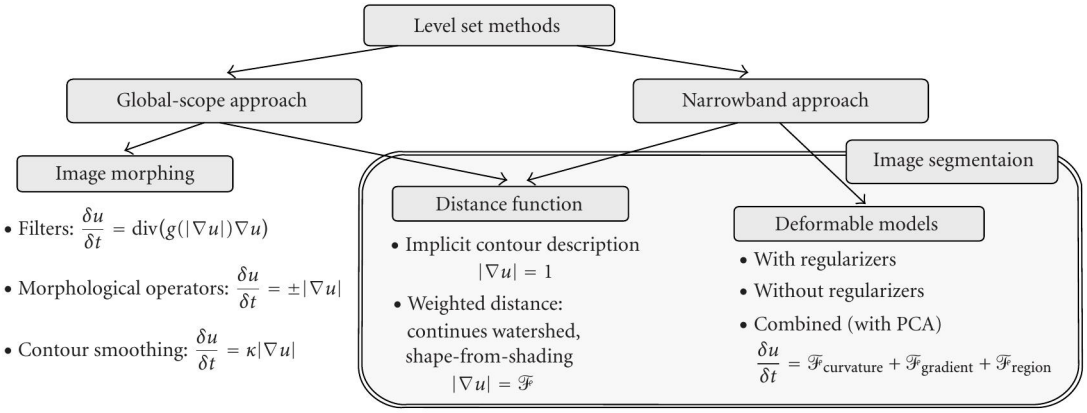


FIGURE 9.1 – Classification des techniques des ensembles de niveau selon le motif de calcul.

Notre classification des techniques des ensembles de niveaux met en évidence un motif unique de calcul. Nous avons deux types de déroulement des itérations (Fig. 9.2) :

i) *itération globale* : on fait évoluer tous les pixels de l'image selon le même schéma numérique. Théoriquement, tous les pixels peuvent être mis à jour en parallèle.

ii) *itération locale* : elle fait évoluer les pixels dans une zone locale ; le plus souvent, elle modélise la position du front à propager selon un schéma numérique. Le plus souvent, il existe un mécanisme limitant le degré de la parallélisation des calculs.

Si on peut constater que les schémas numériques se ressemblent dans les deux types d'itérations, la difficulté majeure consiste en différents types d'accès mémoire pour chaque classe d'algorithmes ; accès mémoire aléatoire pour l'itération locale et accès séquentiel pour l'itération globale.

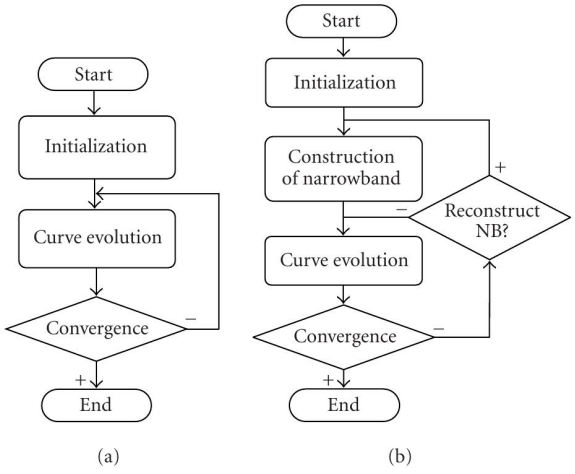


FIGURE 9.2 – Deux itérations : (a) itération globale : image entière ; (b) itération locale : NB - Narrow Band définissant la zone de traitement

Dans les deux types d'itérations, la fonction distance joue un rôle majeur. Par conséquent, proposer une architecture reposant sur l'algorithme Massive Marching permet de débloquer la problématique du tri des pixels, simplifier les accès aux données à maximum et ainsi augmenter des possibilités de parallélisation.

9.2 Architecture matérielle dédiée aux méthodes à base des équations aux dérivées partielles

L'architecture globale présentée sur la Fig. 9.3 repose sur une mémoire partagée à accès non uniformes (NUMA [2]). Les unités de traitement (PU) sont équipées pour réaliser les trois étapes : initialisation, calcul du schéma numérique, mis à jours des valeurs des pixels. Les calculs sont réalisés en virgule fixe, le problème de précision de calcul est détaillé en [53].

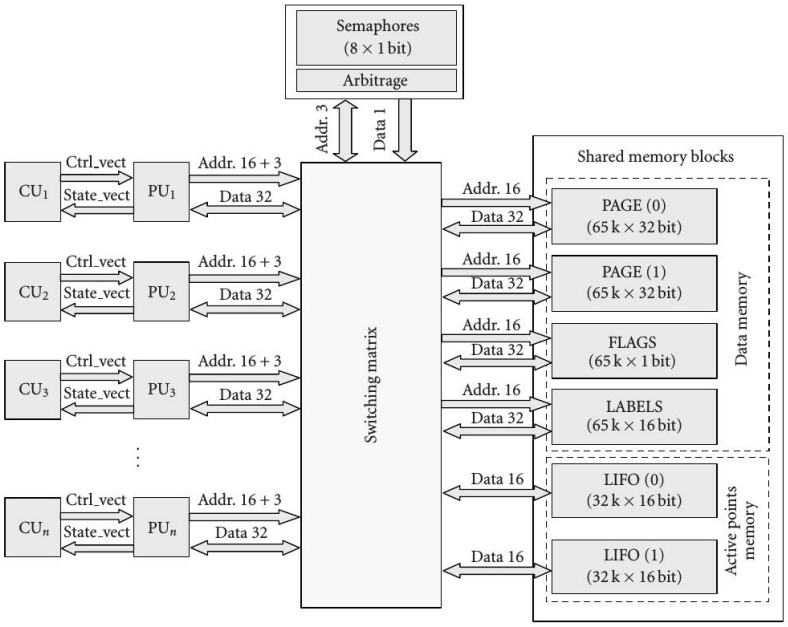


FIGURE 9.3 – Architecture globale pour les méthodes à base des équations aux dérivées partielles.

Notre proposition met en œuvre le principe de calcul SPMD [3]. Les unités de traitement réalisent les mêmes calculs, mais d'une manière asynchrone sur l'ensemble des pixels à traiter. Cependant, il existe des instants de synchronisation au niveau temporel à la fin de chaque étape et de chaque itération pour garantir la cohérence temporelle des données.

Une analyse détaillée du timing de l'exécution asynchrone a été publiée dans [53] et elle est illustrée sur la Fig. 9.4. Elle intègre également l'analyse des délais des temps d'attente induits par les accès à la mémoire partagée et par l'arbitrage.

Les performances de l'architecture proposée ont été évaluées sur deux cas. Le premier est le calcul d'une distance pondérée selon l'algorithme Massive Marching (Fig. 9.5(a)). Il représente le calcul d'une transformée de distance pondérée avec propagation simultanée des étiquettes de régions, il sert à vérifier l'uniformité du flux de données et la répartition de la charge de calcul sur toutes les unités de traitement.

Le deuxième cas implémente un algorithme de suivi d'objet basé sur le contour (Fig. 9.5(b) - 9.5(c)). Il met en œuvre un algorithme de suivi d'objet par contours actifs. Le but de ce test est d'évaluer la capacité de cette plate-forme à fonctionner dans des applications réelles avec

2. NUMA - Non Uniform Memory Access
3. SPMD - Single Program on Multiple Data

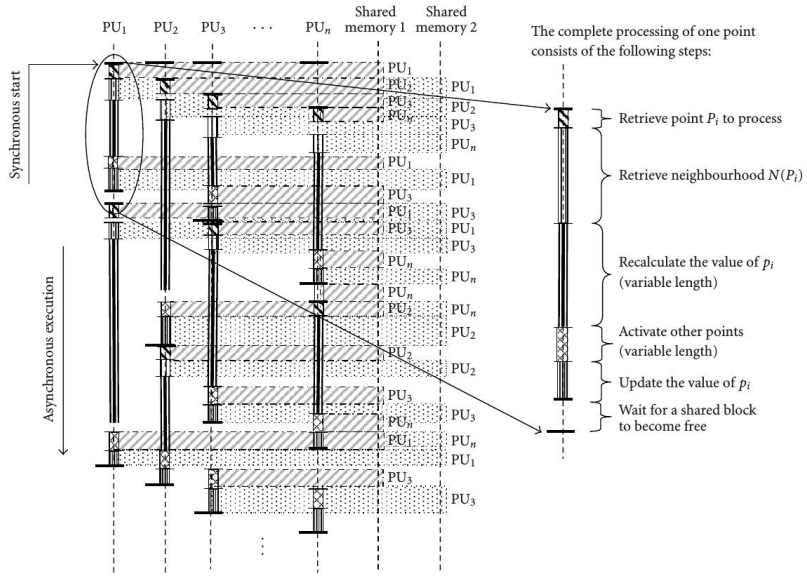
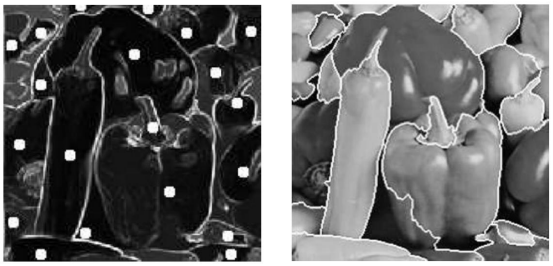


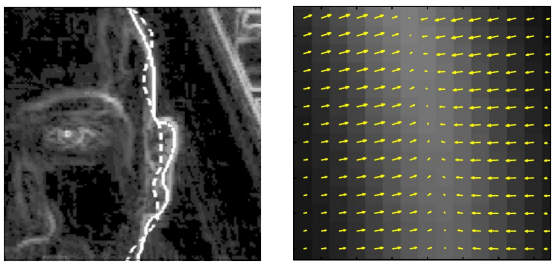
FIGURE 9.4 – Illustration du fonctionnement asynchrone des unités de calcul (PU).

des accès intensifs à la mémoire. Les résultats de la simulation montrent que l'application de suivi de contour peut être exécutée sur notre architecture en temps réel, à condition que les processeurs soient cadencés à 120 MHz et qu'une instruction s'exécute en un cycle d'horloge.

Les résultats des évaluations montrent une augmentation linéaire des performances par rapport au nombre d'unités de traitement et une activité équilibrée (au moins jusqu'à quatre



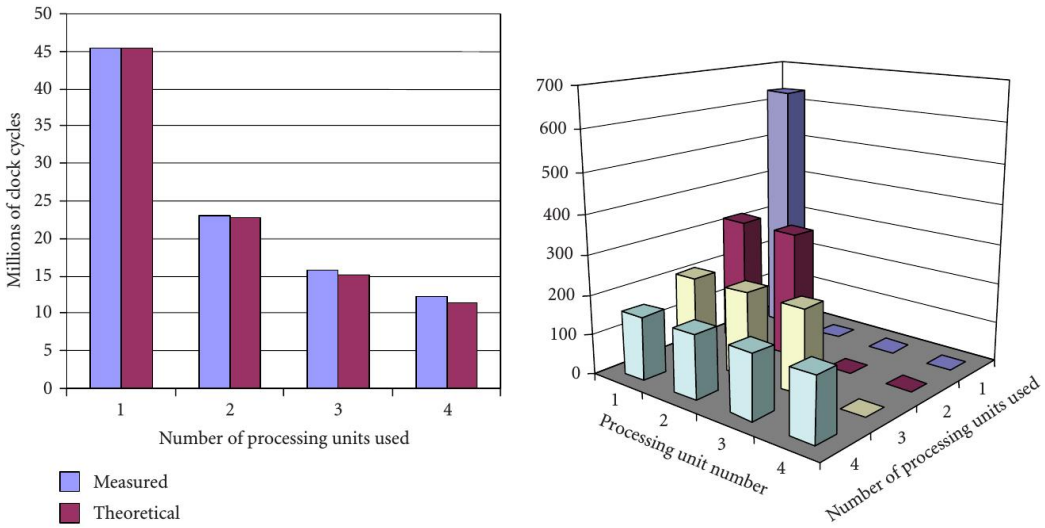
(a) Image de test utilisée pour le calcul de la ligne de partage des eaux continue. A gauche : l'image d'entrée avec des marqueurs, à droite le résultat.



(b) Illustration des phases de l'application de suivi de visage. A gauche : l'évolution de la courbe englobante le visage, à droite : le champs de force d'attraction à base du gradient.

(c) Résultat du suivi de visage à l'aide des contours actifs.

FIGURE 9.5 – Illustration des résultats obtenus sur l'architecture proposée.



(a) Temps de calculs(en nombre de cycles d'horloge).

(b) Répartition de charge entre les unités de traitement.

FIGURE 9.6 – Evaluation des performances de l'architecture par rapport au nombre d'unités de traitement.

unités de traitement fonctionnant indépendamment). L'analyse des performances en fonction du nombre d'unités de traitement est illustrée sur la Fig. 9.6.

En principe, l'architecture dispose d'un certain niveau de scalabilité qui consiste à répliquer les unités de traitement. Physiquement, leur nombre est limité par le silicium disponible sur la puce ; et logiquement, par l'équilibre des flux de données sur tous les blocs de l'architecture.

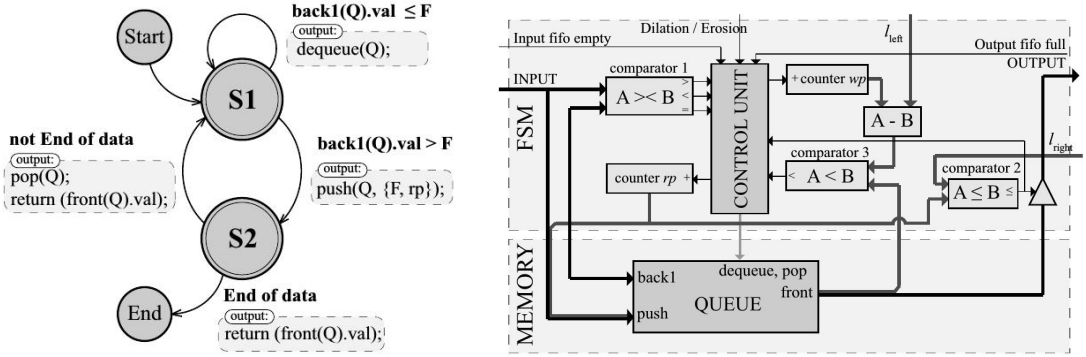
Chapitre 10

Architectures matérielles pour filtres morphologiques concaténés et de grandes tailles

Dans ce chapitre, nous présentons notre contribution à la mise en œuvre matérielle efficace et programmable des algorithmes introduits dans le Chapitre 8. Celle-ci est toujours développée dans le cadre de la collaboration avec Mines-ParisTech et encadrement de thèse de Jan Bartovsky. Nous disposons d'un cadre algorithmique unifié reposant sur des opérateurs morphologiques atomiques 1-D à latence nulle et complexité linéaire. Nous pouvons en construire des opérateurs concaténés sans mémorisation intermédiaire. Nous disposons également de l'ouverture 1-D à l'angle arbitraire avec le calcul simultané des granulométries.

10.1 Architecture pour dilations rectangulaires à flot de données

Nous savons que le cadre algorithmique utilisé pour la dilatation et l'érosion, ainsi que pour l'ouverture par élimination des sommets, est basé sur la file d'attente ; cela signifie que l'implémentation dédiée doit traiter majoritairement les opérations sur la mémoire représentant la file d'attente (FIFO). Si nous représentons l'algorithme de dilatation 1-D sous forme de d'une machine à états finie (FSM [1]), nous pouvons voir que l'algorithme est constitué de seulement 2 états principaux {S1, S2} (Fig. 10.1).

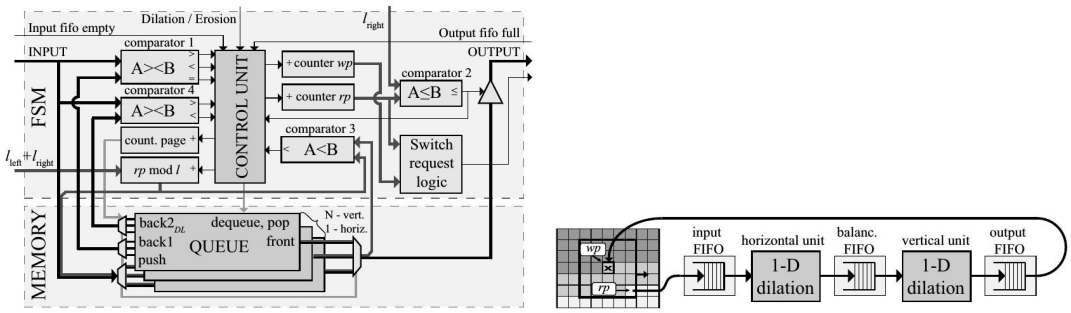


(a) FSM de l'algorithme de dilatation 1-D.

(b) Architecture matérielle de la dilatation horizontale 1-D.

FIGURE 10.1 – Implémentation matérielle de l'algorithme de dilatation 1-D

1. FSM - Final State Machine)



(a) Architecture de la dilatation verticale.

(b) Concaténation de la dilatation horizontale et verticale pour obtenir une dilatation 2-D.

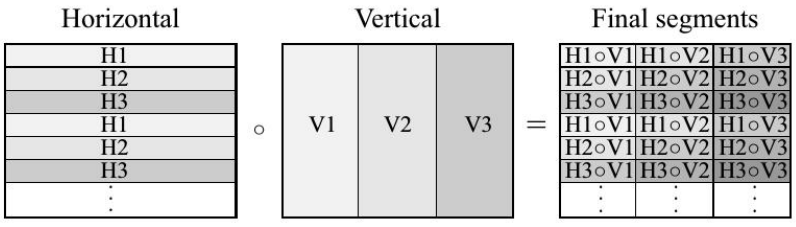
FIGURE 10.2 – Architecture matérielle pour opérateurs morphologiques 2-D à SE rectangulaire

L'état S1 représente l'étape analysant les données d'entrée. C'est la partie dépendante des données. En conséquence, son temps de calcul varie sans toutefois dépasser 1 à l cycles d'horloge (l désigne la longueur du SE) dans le pire des cas. L'état S2 est essentiellement dédié à la phase de la suppression des valeurs inutiles. Ainsi la FSM gère la totalité de la procédure de calcul et stocke temporairement les valeurs dans la partie de mémoire.

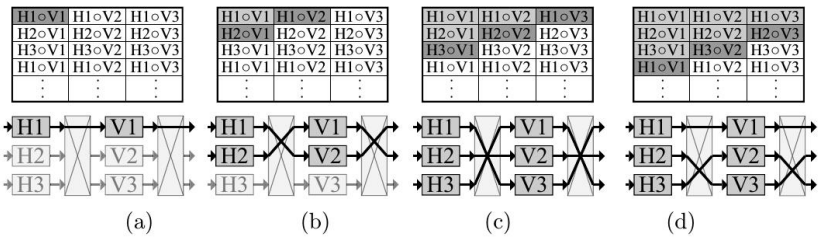
L'architecture supportant le SE vertical reprend les mêmes principes. La distinction principale est que l'architecture verticale a besoin de N files d'attente (Queue). Il s'agit d'un ensemble de files d'attente parallèles et indépendantes dont la taille ne dépasse pas la taille du SE maximal autorisé (Fig. 10.2(a)).

La dilatation par un rectangle peut être implémentée comme une concaténation de deux blocs de dilatation 1-D travaillant avec des SE horizontaux et verticaux (Fig. 10.2(b)). Les performances sont encore améliorées par l'introduction du parallélisme de calcul.

Nous proposons une solution permettant de définir un degré de parallélisme arbitraire car une distribution de calcul déterministe est possible. Le principe est basé sur un partitionnement de l'image d'entrée illustré sur la Fig. 10.3.



(a) Partitionnement de l'image pour les calculs parallèles.



(b) Illustration du flux de données pour des dilatations parallèles et la distribution des calculs.

FIGURE 10.3 – Illustration du principe du parallélisme pour dilatation rectangulaire. H_i - dilatation horizonale i, V_i - dilatation verticale i.

Les résultats d'évaluation des performances sur une cible matérielle type FPGA Xilinx Virtex6 sont résumés sur la Fig. 10.4. Nous pouvons constater que la latence initiale de calcul est proportionnelle à la taille de l'élément structurant, mais n'est en aucun cas, perturbée par la concaténation des opérateurs ou par la taille de l'image. Les résultats obtenus pour les différents degrés de parallélisme confirment ces hypothèses. Le speed-up obtenu expérimentalement augmente linéairement avec le degré de parallélisme.

Size of SE (square)	3x3	11x11	21x21	31x31	41x41
Latency [image line]	1	5	10	15	20
$PR_{\text{experimental}}$ [clk/px]	2.344	2.379	2.409	2.440	2.470
$PR_{\text{compensated}}$ [clk/px]	2.338	2.350	2.350	2.352	2.353
Registers	212	232	242	242	252
LUTs	584	761	859	859	953
Block RAMs	2	6	13	13	28

(a) Latence de calcul par rapport à la taille de l'élément structurant.

Size of image	CIF	VGA	SVGA	XGA	1080p
Latency [image line]	15	15	15	15	15
$PR_{\text{experimental}}$ [clk/px]	2.569	2.484	2.440	2.404	2.391
$PR_{\text{compensated}}$ [clk/px]	2.381	2.375	2.352	2.339	2.348
FPS [frame/s]	384	130	85	51	20.5
Registers	231	237	242	242	253
LUTs	761	853	859	859	1057
Block RAMs	7	13	13	13	26

(b) Latence de calcul par rapport à la taille de l'image d'entrée.

FIGURE 10.4 – Évaluation des performances de l'architecture à flot de données pour dilatation 2-D .

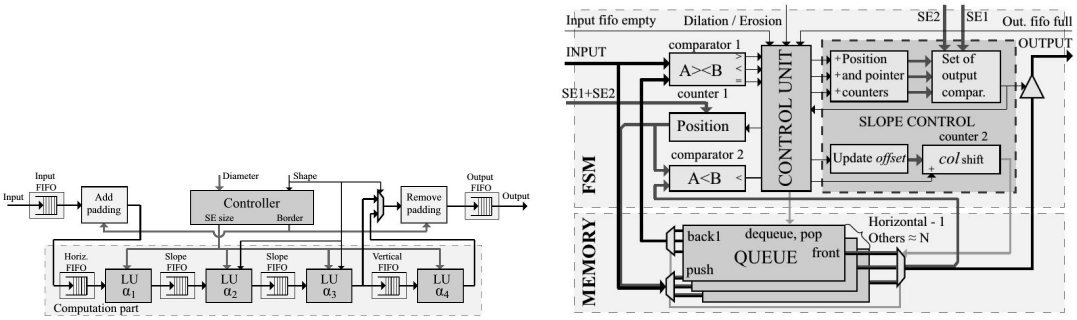
Parallelism degree *PD*	1	2	3	4	5	6
Latency [image line]	15	15	15	15	15	15
$PR_{\text{experimental}}$ [clk/px]	2.440	1.264	0.824	0.625	0.505	0.426
Experimental speed-up [-]	1	1.930	2.858	3.770	4.663	5.532

FIGURE 10.5 – Évaluation des performances par rapport au degré de parallélisme.

10.2 Architecture matérielle de dilatation 2-D polygonale

Les opérations à base des SE rectangulaires ont une propriété désagréable pour le traitement d'images : l'anisotropie angulaire. Dans beaucoup d'applications, on préfère utiliser des SE isotropes qui affectent l'image de manière égale dans toutes les directions. Parce que les cercles avec un diamètre programmable sont très difficiles à mettre en œuvre efficacement, on utilise l'approximation par des polygones réguliers qui peuvent être décomposés en lignes orientées.

Par conséquent, une architecture matérielle supportant les éléments structurants polygonaux pourrait être très intéressante. Pour la concevoir, nous devons enrichir le bloc de dilatation 1-D décrit dans la section précédente, qui a été conçu pour l'orientation horizontale et



(a) Concaténation des opérateurs pour les éléments structurants polygonaux.

(b) Unité de traitement LU_{α_i}.

FIGURE 10.6 – Architecture matérielle pour la réalisation d'un élément structurant polygonal.

verticale seulement, pour pouvoir orienter le SE sous les angles arbitraires, par exemple α_i = 45, 60, 120, 135 degrés pour hexagone. Nous appelons cette unité de calcul l'unité de ligne (LU_{α_i}) orientée sous un angle α_i.

L'architecture de l'unité LU_{α_i} capable de dilatation par des segments orientés est illustrée sur la Fig. 10.6(e). On peut la voir comment une unité de calcul de la dilatation verticale complétée par l'unité Slope contrôle. Cette dernière, elle-même est basée sur l'algorithme de Bresenham. Par la suite, une concaténation appelée Unité Polygone (PU) des unités LU_{α_i} permet de réaliser un SE polygonal complet par décomposition.

Les propriétés de cette implémentation sont validées par les expérimentations résumées sur la Fig. 10.7. Également, pour les SE polygonaux, nous mettons en la parallélisation par partitionnement de l'image et grâce aux bonnes propriétés des algorithmes dès le départ, nous obtenons des performances très satisfaisantes.

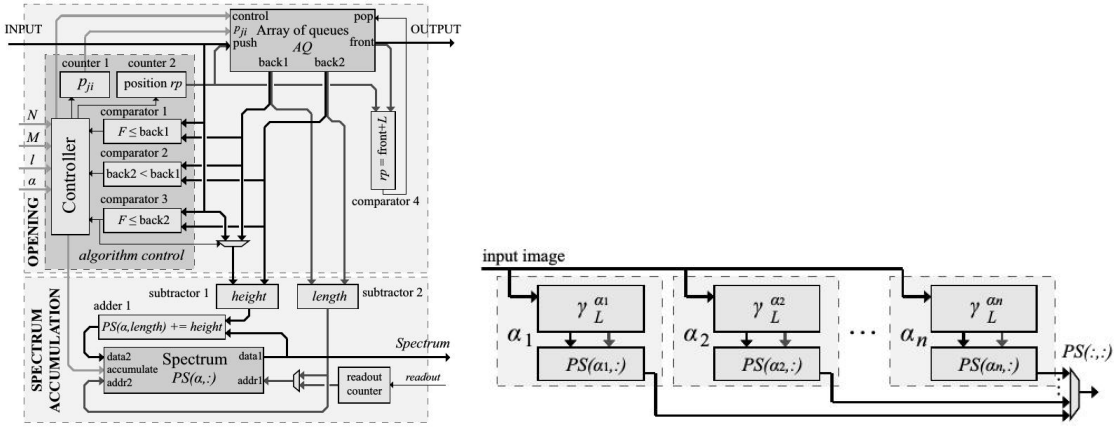
Image Size	VGA	SVGA	XGA	1080p
$PR_{\text{experimental}}$ (PU) [clk/px]	2.61	2.53	2.53	2.44
Latency [image line]	25	25	25	25
FPS (PU) [frame/s]	125	82	50	19
FPS (PPU) [frame/s]	599	406	257	105
Experimental speed-up [-]	4.79	4.94	5.1	5.34

(a)

Parallelism degree *PD*	2	3	4	5	6
FPS [frame/s]	162	234	306	376	441
Experimental speed-up [-]	1.92	2.77	3.62	4.44	5.22

(b)

FIGURE 10.7 – Évaluation de performances des l'architectures pour les éléments structurants polygonaux.



(a) Architecture de l'ouverture 1-D à angle arbitraire.

(b) Concaténation des unités de calculs pour le calcul de la distribution de taille.

FIGURE 10.8 – Opérateur de l'ouverture 1-D et le calcul simultané des granulométries.

10.3 Ouverture morphologique 1-D et le calcul simultané des granulométries.

Cette section décrit l'implémentation matérielle de l'algorithme le d'ouverture 1-D par l'élimination des sommets. L'architecture proposée calcule à la fois l'ouverture à l'angle arbitraire et les granulométries en flot de données. On garde toujours la même démarche.

Encore ici, c'est une machine à états finis implémentant le comportement conditionnel de l'algorithme (while, if) et l'ordonnancement des commandes de la même manière que l'algorithme de dilatation.

L'algorithme utilisé supporte l'orientation arbitraire du SE. A nouveau, l'approche consiste à diviser une image en couloirs parallèles inclinés et à laisser à chaque couloir avoir sa propre file d'attente en mémoire. Comme l'algorithme lit les données séquentiellement, dans l'ordre de balayage horizontal, il ne reste qu'à déterminer à quel couloir appartient le pixel en cours de traitement. Alors la tâche de l'orientation SE se réduit au simple calcul du pointeur de sélection des couloirs. Également ici, nous proposons une implémentation parallèle scalable et déterministe. Ces particularités sont détaillées dans [69].

La lecture de l'analyse des performances (Fig. 10.9) est plus complexe dans le cas des polygones. La latence de calcul est influencé par la taille et par l'angle de rotation de l'élément structurant. Cependant, cette implémentation reste toujours celle qui produit la latence minimale par rapport à l'état de l'art. Son évaluation théorique est démontrée dans [19].

Length of SE	15	31	63	127	255
Latency [image lines]	10	21	44	89	180
$PR_{\text{experimental}}$ [clk/px]	2.350	2.453	2.602	2.900	3.485
$PR_{\text{compensated}}$ [clk/px]	2.287	2.323	2.334	2.343	2.298
FPS [frame/s]	88	85	80	72	60
Registers	321	350	395	473	473
LUTs	1612	1643	1734	1840	2052
Block RAM	6	13	28	60	128

(a)

Orientation α [°]	0	15	30	45	60	75	90
FPS [frame/s]	84.6	81.7	80.2	79.4	79.35	79.8	81.2
Latency [image lines]	0	16	31	44	54	60	62

(b)

FIGURE 10.9 – Évaluation de performances de l'architecture pour les ouvertures à élément structurant linéaire avec angle arbitraire (Xilinx FPGA Virtex6).

10.4 Co-processeur morphologique pipeliné

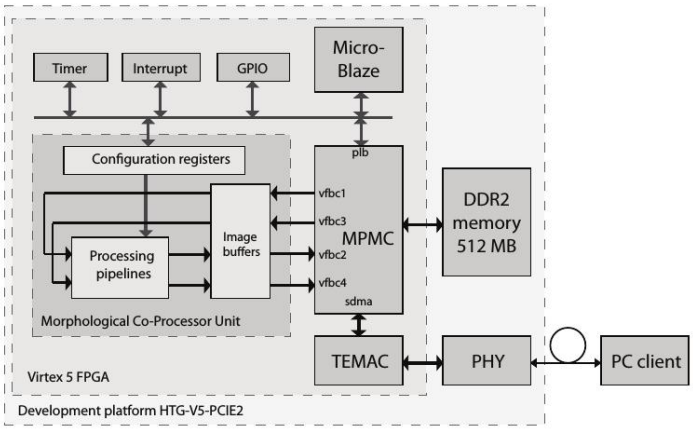
Les résultats présentés ici proviennent d'une intégration des solutions précédentes. Ils sont le résultat de l'étude applicative de fin de thèse au LIGM et de la période post-doctorale de Jan Bartovsky au Centre de Morphologie Mathématique que j'ai pu co-encadrer. Ils démontrent comment nous proposons de construire autour des pipelines des opérateurs individuels une plateforme de calcul avancée, polyvalente avec une interface facilitant son utilisation. L'ensemble des résultats de cette partie a été publié dans [1].

Nous avons enrichi l'ensemble des opérateurs supportés en ajoutant des unités pour des opérateurs géodésiques, des unités arithmétiques-logiques (ALU), un contrôleur de mémoire DDR2 rapide et un lien Ethernet rapide pour transférer des images. Le calcul est contrôlé et surveillé par un microcontrôleur Xilinx MicroBlaze qui assure également la communication avec le monde extérieur. La plate-forme entière se comporte comme un serveur et répond aux demandes de calcul du client. Une interface logicielle fournie (en python et en C C ++) offre à l'utilisateur une possibilité pratique d'interagir avec la plate-forme.

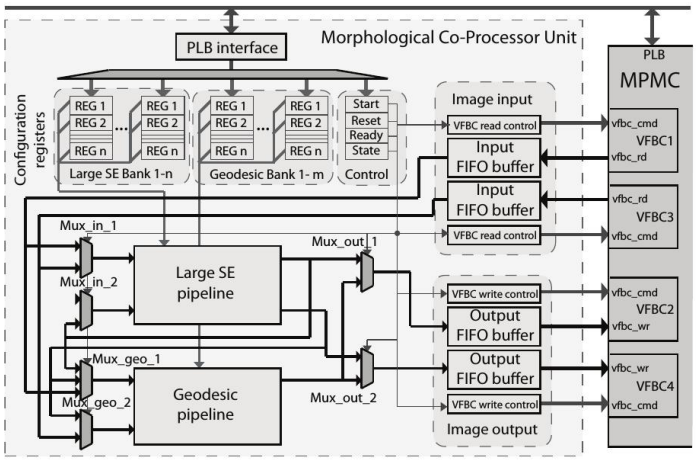
Afin de gagner en flexibilité, nous proposons deux pipelines parallèles d'unités programmables pour SE de grande taille. Pour exécuter efficacement la plus grande collection d'opérateurs (y compris ceux avec deux branches parallèles, telles que chapeau haut de forme ou gradient) les pipelines sont interconnectés par des ALUs. La sortie de l'ALU peut être routée vers l'un des ports d'entrée de l'étape suivante (ou les deux). Cette capacité de routage de flux augmente l'adaptabilité de l'architecture pour exécuter différents algorithmes.

Le schéma d'interconnexions des pipelines est pensé pour une polyvalence maximale. L'architecture peut être programmée pour réaliser les calculs suivants : gradient, ouverture, fermeture ou ses résidus, filtres séquentiels ou granulométries, reconstruction morphologique, ouvertures et fermetures par reconstruction. Cette variété d'opérateurs permet d'apprécier pleinement le potentiel de cette plate-forme offrant des pipelines flexibles. Enfin, les deux pipelines peuvent également être utilisés indépendamment et en parallèle sur les mêmes données d'entrée.

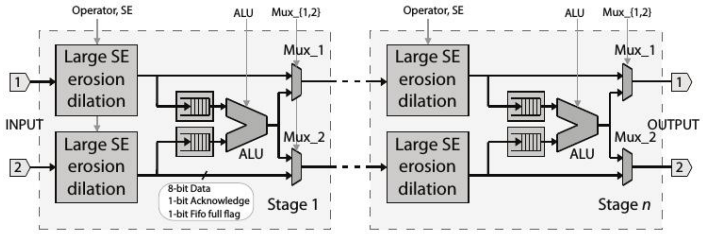
La plate-forme est accessible via une interface Ethernet à trois vitesses utilisant les protocoles TCP IP ou UDP IP. Le MCPU exécute un serveur capable d'accepter des images et



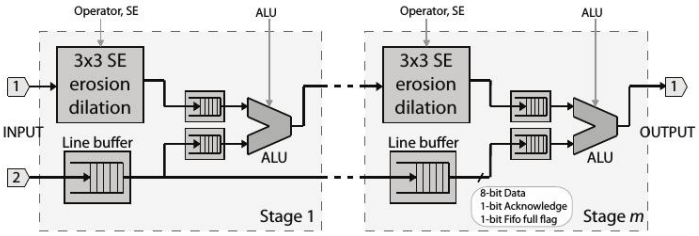
(a) Plateforme complète.



(b) Architecture globale du coprocesseur morphologique.



(c) Pipeline dédiée aux opérateurs morphologiques de grandes tailles.



(d) Pipeline dédiée aux opérateurs morphologiques géodésiques.

FIGURE 10.10 – Co-processeur morphologique.

des opérations à exécuter via le lien Ethernet d'un client supérieur.

Nous comparons les performances de l'architecture proposée avec une autre solution

Operator	Shape of SE	Size of SE or λ	MCPU	OpenCV at Sabre	Smil at Sabre	OpenCV at Xeon
Dilation	Rectangle	3×3	21.9	32.7	8.4	0.58
Opening	Rectangle	151×151	24.3	2450	1083	38.6
Opening	Octagon	151×151	41.9	246 s	2453	2301
Opening by recon.	Rectangle	151×151	544	47.6 s	22.1 s / 2110*	1940
Opening by recon.	Octagon	151×151	512	289 s	21.1 s	4356
ASF	Rectangle	$\lambda = 11$	64.2	4530	1987	57.1
ASF	Octagon	$\lambda = 11$	83.3	77 s	3872	814
Pattern spectrum *PS*	Rectangle	$\lambda = 11$	62.3	2570	1098	53.8
Pattern spectrum *PS*	Octagon	$\lambda = 11$	62.7	21.2 s	1782	249
PS by recon.	Rectangle	$\lambda = 11$	2530	190 s	85.3 s / 18.2 s*	8920
PS by recon.	Octagon	$\lambda = 11$	2410	201 s	81.5 s	8751

FIGURE 10.11 – Résultats de performance des opérateurs sélectionnés. L'image est une photo naturelle de 1000 × 1000 px, les résultats sont en millisecondes (sauf si les secondes sont spécifiées).

	Processing unit				Hardware System		Application Example ASF[6]		
	Technology	Supported SE	Throughput [Mpx/s]	f_{max} [MHz]	Number of units	Supported image	Image scans	FPS [frame/s]	Latency [px]
Clienti [4]	FPGA	arbitrary 3×3	403	100	16	1024×1024	6	66.7	$5NM + 84N$
Chien [3]	ASIC	disc 5×5	190	200	1	720×480	45	12.2	$44NM + 84N$
Déforges (a) [8]	FPGA	arbitrary 8-convex	50	50	1	512×512	13	14.7	$12NM + 84N$
Déforges (b) [8]	FPGA	arbitrary 8-convex	50	50	13	512×512	1	50	$84N$
This paper	FPGA	regular polygon	195	100	13	1024×1024	1	185	$84N$

FIGURE 10.12 – Comparaison avec des propriétés des architectures efficaces existantes ([1]). N - largeur de l'image, M-hauteur de l'image.

embarquée et pleinement programmable - un processeur ARM (Fig. 10.12. Dans notre cas, nous utilisons la plate-forme Sabre IMX.6 de Freescale, qui peut être considérée comme un autre exemple de plate-forme portative. La plate-forme Sabre utilise un processeur quad-core ARM A9 à 1 GHz, 1 Go de la mémoire DDR3 jusqu'à 533MHz, et fonctionne sous Linux avec la pile TCP IP. Nous avons créé des benchmarks pour deux bibliothèques de traitement d'images : (i) le célèbre OpenCV, et (ii) Smil hautement optimisé. Dans la comparaison, nous avons également inclus les résultats single-thread de l'OpenCV sur PC de bureau Intel Xeon E5620, 2,40 GHz, avec 24 Go de mémoire, fonctionnant sous Fedora, version 20, Linux.

Également, nous avons effectué une comparaison de notre architecture avec quelques autres qui supportent le SE plat et non rectangulaire. On peut constater que la solution de Clienti [87] donne un débit élevé pour un SE élémentaire 3 × 3. La proposition de Chien [86] à base d'un ASIC atteint un débit raisonnable avec une petite SE 5 × 5 de forme diamant. L'architecture proposée par Déforges [94] supporte des SE 8-convexes qui se décomposent en une concaténation de SE élémentaire de 2 pixels. (Un débit inférieur est probablement dû à un dispositif moins puissant que celui des autres implémentations.) Pour toutes les autres architectures, la flexibilité pour contrôler la taille et la forme du SE après la synthèse reste non précisée.

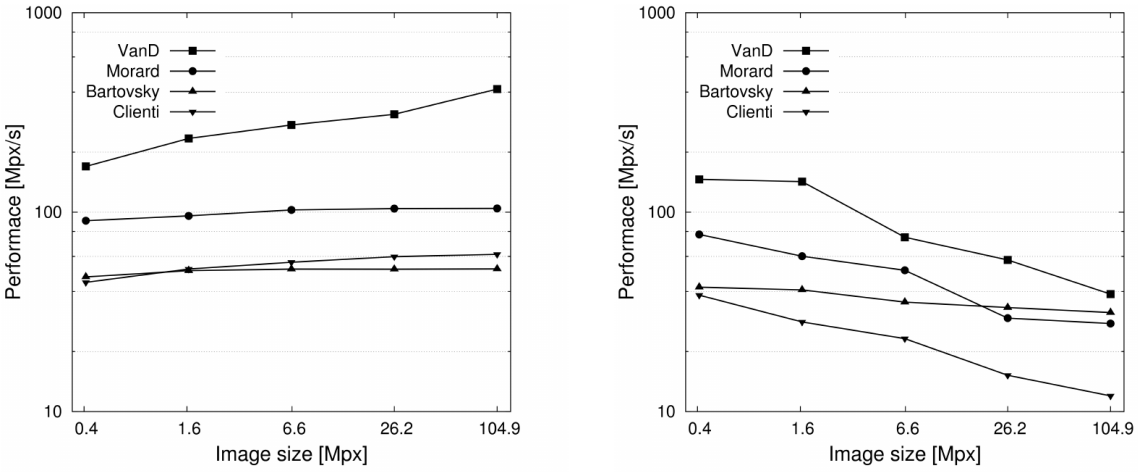
Notons que dans le contexte des éléments structurant de grande taille, toutes les architectures concurrentes doivent faire appel au principe de l'homothétie pour obtenir des SE plus grands. Cela nécessite d'utiliser un long pipeline de traitement comme dans Clienti ou Déforge et al. Cela diminue considérablement l'ensemble débit et accentue davantage l'apport de notre solution.

10.5 Implantation concurrente sur GPU

Nous avons déjà présenté les ouvertures et les fermetures morphologiques linéaires qui sont d'importants opérateurs non linéaires de la morphologie mathématique. Dans les appli-

cations pratiques, de nombreuses orientations différentes des segments de ligne numériques doivent généralement être prises en compte. C'est pourquoi il nous a paru particulièrement intéressant d'étudier une implantation concurrente logicielle de notre cadre algorithmique dédié aux opérateurs morphologiques d'ouverture et de fermeture de grande taille, sous angle arbitraire en une seule passe.

Pendant le stage de mobilité doctorale de Pavel Karas de l'Université Masaryk de Brno, nous avons d'abord analysé des algorithmes séquentiels et parallèles efficaces pour le calcul des ouvertures et des fermetures linéaires, nous avons réalisé les comparaisons des performances des implémentations CPU de quatre meilleurs algorithmes proposés par Van Droogenbroeck [99], Morard [141], Clienti [87], Bartovsky [52]. Sur la Fig. 10.13, nous pouvons constater qu'uniquement l'algorithme de Bartovsky est peu influencé par l'orientation du SE.



(a) Horizontal SE

(b) Vertical SE

FIGURE 10.13 – Comparaison des implémentations de CPU pour les SE horizontales et verticales de taille env. 5% de la largeur de l'image.

C'est pourquoi nous avons proposé son implémentation sur un GPU. Nous avons évalué ses performances sur des images réelles, dans le cas des diverses applications [7].

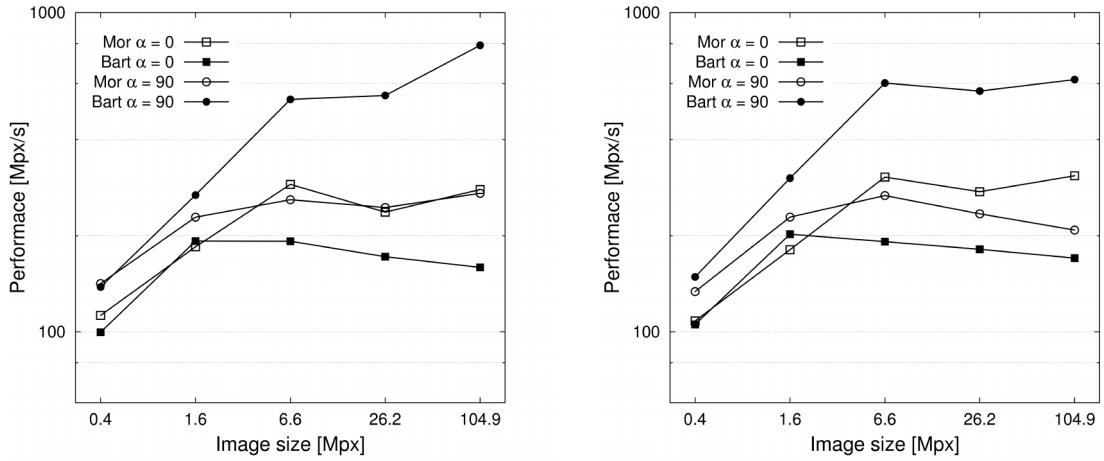
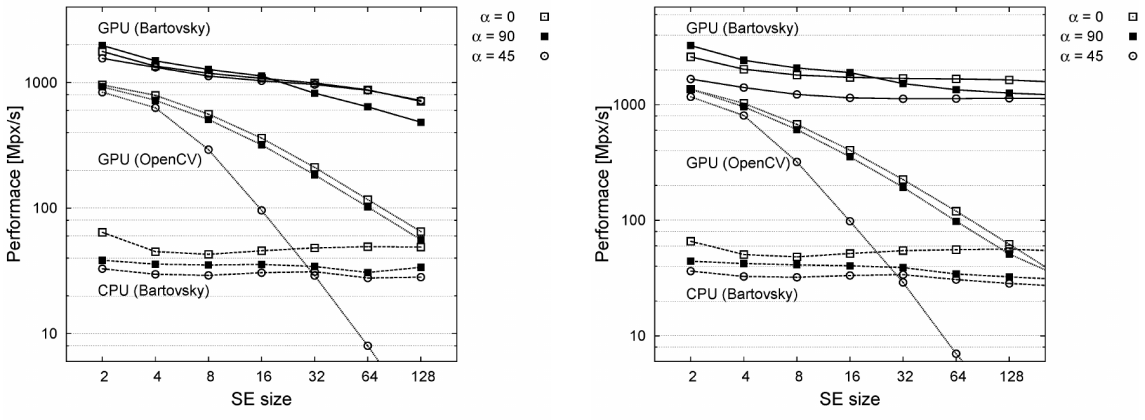


FIGURE 10.14 – Comparaison des implémentations de GPU de l'algorithme de Morard et Bartovsky pour les SE horizontaux et verticaux de taille env. 5% de la largeur de l'image.

D'après nos résultats expérimentaux, il est apparu que la mise en œuvre proposée du GPU était adaptée aux applications comportant de grandes images industrielles, fonctionnant sous

des contraintes sévères de temps. Les résultats obtenus dépassent également en performances les librairies de traitement d'image de référence telles que OpenCV, version GPU.



(a) Image D15, 1280 × 1280 px

(b) Image D15, 5120 × 5120 px

FIGURE 10.15 – Comparaison des implantations CPU et GPU de l'algorithme de Bartovsky et OpenCV pour les SE inclinés, de taille env. 5% de la largeur de l'image.

Chapitre 11

Arbre des composantes connexes

Mes travaux ont toujours démontré l'intérêt d'étudier des approches algorithmiques unifiantes. C'est-à-dire des cadres algorithmiques permettant d'unifier le formalisme mathématique et d'adresser à la fois des traitements d'images bas niveau (niveau pixellique) ainsi que de haut niveau (niveau des régions ou des objets).

Autrement, l'optimisation des implémentations, visant des applications complètes allant du bas niveau jusqu'au haut niveau de traitement, devient particulièrement complexe au vu des contraintes qui peuvent être orthogonales l'une vers des autres : structures de données différentes, modèles mathématiques différents et précision de calcul différente.

Comment nous l'avons déjà mentionné à plusieurs reprises, le problème d'efficacité d'implémentation est étroitement lié aux structures de données utilisées sur lesquelles t'appuie le traitement d'images en question.

De ce point de vue, les algorithmes de traitement d'images à base de graphes semblent être très intéressants, car ils permettent de combler le fossé de représentation des données pour tous ses niveaux d'abstraction.

Parmi des approches à base de graphes, nous nous intéressons aux méthodes à base de l'arbre des composantes connexes (CCT - Connected Component Tree). En effet, le CCT est une structure unique de données qui peut être appliquée à tous les niveaux de traitement d'images : du filtrage au niveau pixellique à l'analyse d'image.

Pour illustrer la richesse de ce formalisme, citons quelques exemples d'applications. Nous pouvons citer Salembier [155], ou Wilkinson et al. [176], ainsi que l'analyse d'image : extraction de mouvement présentée en [155], segmentation [90], [133], reconnaissance de formes en imagerie astronomique [71], en applications industrielles [74], analyse d'images microscopiques [92], applications de vidéo-surveillance [150], remote sensing [104] ou visualisation de données [85].

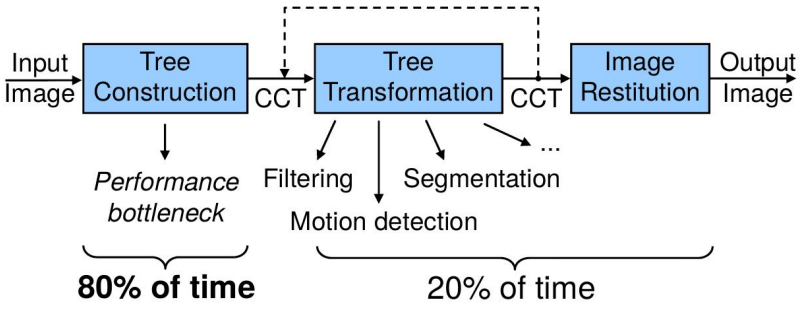


FIGURE 11.1 – Étapes des applications à base de CCT.

Les étapes typiques d'une application basée sur le CCT sont illustrées sur la Fig. 11.1.

Une fois que le CCT est construit, les opérateurs de traitement d'images sont réalisés par les transformations de graphe. Ainsi, une seule représentation de données est utilisée pour l'ensemble de l'application. Pendant ou après la construction du CCT les attributs des composants connexes sont calculés. Ils permettent la caractérisation et la quantification de nombreuses propriétés de composants, y compris (mais sans s'y limiter) la forme, qui qualifie le CCT en partie comme une représentation géométrique. Enfin, comme les composants sont organisés en une structure hiérarchique créant ainsi un modèle relationnel simple entre les composantes connexes.

Les expériences montrent que le principal goulot d'étranglement est la construction elle-même du CCT, consommant environ 80% du temps d'exécution [8] de l'application. En revanche, une fois que le CCT a été construit, le reste de l'application s'exécute assez rapidement. Dans les parties suivantes nous décrivons notre contribution à l'accélération de la construction du CCT. Ces travaux ont été l'objet de thèse de Petr Matas [130].

Considérations pratiques

Représentations de l'arbre des composantes connexes

Le concept-clé utilisé par les algorithmes à base de CCT est la notion de connexité (Figure 11.2) qui permet de définir l'objet de composante connexe.

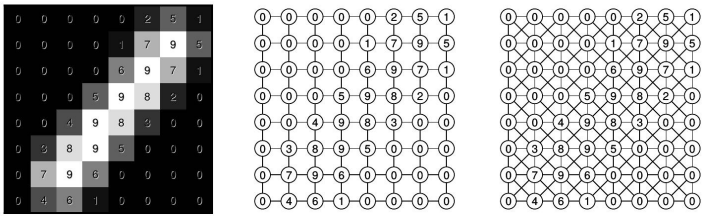


FIGURE 11.2 – Exemples de connexité ; de gauche à droite : exemple d'une image, l'exemple de 4-connexité, l'exemple de 8-connexité [130]

Dans le cas binaire, une composante connexe est un sous-ensemble connexe maximal des pixels de premier plan avec un chemin contigu entre les pixels [176]. Par exemple, l'image binaire illustrée sur la Fig. 11.3 contient deux composantes connexes.

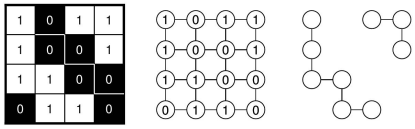


FIGURE 11.3 – Exemple des composantes connexes présentes dans l'image binaire [130]. À gauche : image d'entrée ; à droite : composantes connexes.

Dans le cas d'une image en niveaux de gris, les composantes connexes sont des ensembles connexes de chaque niveau de l'image [176]. Une relation d'inclusion organise tous les composants connexes de l'image en niveaux de gris en une arborescence (Fig. 11.4).

Deux types d'informations sont contenus dans l'arbre des composantes connexes : i) relations parentenfant entre les composantes, ii) Appartenance des pixels aux composantes connexes. En fonction des différentes représentations du CCT (Fig. 11.5), ces informations sont organisées de manière différente dans la mémoire. Les besoins en mémoire pour chaque représentation du CCT sont reportés dans le tableau de la Fig. 11.5. Un cas spécial est l'algorithme 1-D introduit dans [136].

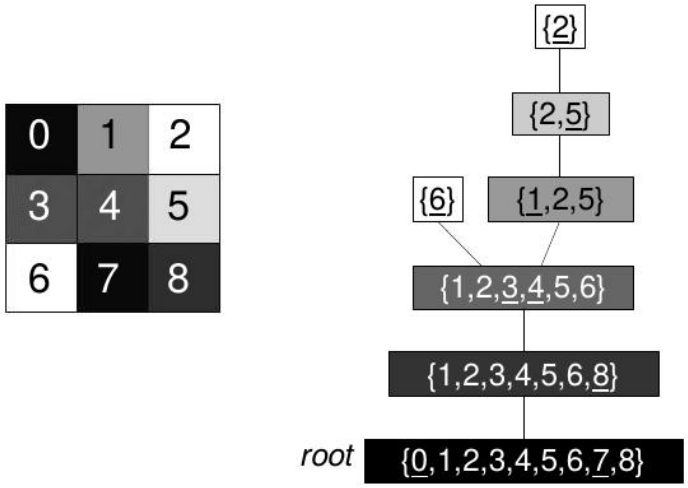
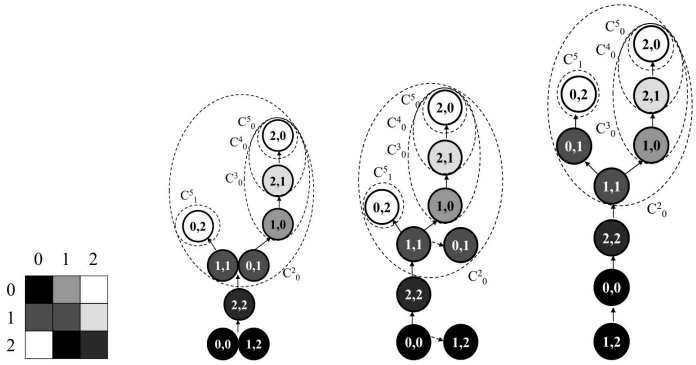


FIGURE 11.4 – Exemple des composantes connexes présentes dans l'image à niveaux de gris [130]



Tree type	Complexity	Data type	Memory requirements
Native CCT [38]	$\mathcal{O}(nG)$	int	$4n$
Canonical CCT [32]	$\mathcal{O}(n\alpha(n))$	int, float	$6n$
Point tree [5]	$\mathcal{O}(n\log(n))$	int, float	$4n$
1-D [26]	$\mathcal{O}(n)$	int, float	n

FIGURE 11.5 – Exemples des différentes structures des CCT. De gauche à droite : image d'entrée, CCT classique, CCT canonique, CCT appelé Point Tree.

Algorithmes de construction des CCT

Trois classes principales d'algorithmes séquentiels existent dans la littérature :

— Algorithmes par inondation : un balayage de l'image d'entrée est requis pour identifier le pixel au plus bas niveau dans l'image, il représentera la racine de l'arbre. Ensuite, un envahissement des composantes connexes par inondation est réalisé. Celui-ci est contrôlé par les valeurs du voisinage. Le processus repose sur l'utilisation de données ordonnées structures : i) files d'attente hiérarchiques, inutilisables pour la représentation des pixels en virgule flottante, ii) files d'attente de priorité, inefficaces du point de vue et de la mémoire.

— Algorithmes par émergence : les composants émergents sont traités comme des ensembles disjoints de pixels, dans l'ordre décroissant. Ensuite, les ensembles disjoints fusionnent pour former un arbre en utilisant l'algorithme union-find [168]. La compression des chemins peut être utilisée pour accélérer l'algorithme et dans certains cas

elle permet de réduire l'empreinte de mémoire.

— Algorithmes par fusion : Ils divisent une image en blocs et calculent l'arbre de composantes sur chaque sous-image. Ces arbres sont ensuite fusionnés pour former l'arbre de l'image entière. Ces algorithmes sont bien adaptés au parallélisme en utilisant une approche divide-and-conquer. Lorsque les blocs sont des lignes d'image, des algorithmes spécifiques en 1-D peuvent être utilisés. L'avantage des algorithmes 1-D est que l'ordre de traitement des pixels devient inutile et l'arbre peut être construit en temps linéaire et très rapidement.

Afin d'améliorer le temps d'exécution, certains efforts de parallélisation ont été faits par le passé. En [135], Meijster étudie une des premières implémentations parallèles de la construction CCT sur des machines à mémoire partagée. L'algorithme est basé sur la fusion d'arbres qui permet la division de l'image en un nombre arbitraire de partitions pour paralléliser la construction CCT. En 2008, Wilkinson propose une amélioration de l'algorithme de Salembier [155] pour la construction de l'arbre complétée par une fusion binaire. Une démonstration de performance de calcul sur des données 3D a été publiée [176].

Dans les travaux de nos doctorants P. Matas et de N. Ngan, nous nous sommes intéressés à la problématique des implémentations efficaces des calculs à base du CCT, adaptées au monde de l'embarqué. Avec Nicolas Ngan, nous avons proposé d'inverser les relations dans la structure Point Tree permettant de simplifier la construction du CCT. Nous appelons cette variante Parent Point Tree. Par la suite, un accélérateur matériel a été proposé pour le calcul d'une application complète. Ce travail d'intégration répond à des besoins d'intégration des algorithmes à base de CCT dans certains produits de SAFRAN Defense.

Également, avec P. Matas, nous avons exploré l'approche par émergence et l'approche par fusion des arbres issus des algorithmes 1-D et ceci sur les calculateurs parallèles et nous explorons également le domaine des implémentations sur FPGA.

11.1 Accélérateurs matériels pour les applications à base du CCT

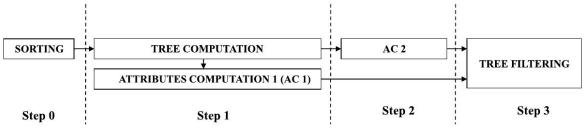
La construction de l'arbre commence que par le tri des pixels sur l'ensemble de l'image, comme illustré à la Fig. 11.6(a). Puis, le calcul de certains attributs et la construction du CCT peuvent être réalisés en parallèle. Par conséquent, les valeurs de pixels accédées pour la construction d'arbre, peuvent être immédiatement réutilisées pour le calcul d'attribut. Cela permet de minimiser les accès à la mémoire et de paralléliser les tâches de calcul.

S'il est possible de démarrer le processus de filtrage de l'arbre parallèlement à la construction de l'arbre en utilisant des attributs (par exemple la hauteur). C'est en fait réalisable, mais il en résulte évidemment des exigences de mémoire plus élevées pour traiter les nombreuses parties incomplètes de l'arbre. En conséquence, il est plus efficace de démarrer le processus de filtrage après la construction complète de l'arbre et le calcul des attributs tel que représenté sur la Fig. 11.6(a).

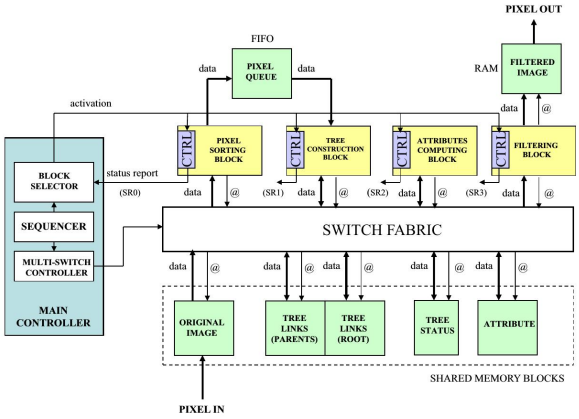
L'architecture globale (voir Fig. 11.6(b)) du système reflète les quatre étapes d'exécution présentées : le tri, la construction d'arbre, le calcul d'attribut et le filtrage.

Cette architecture résulte d'intégration de trois améliorations principales : une proposition de structure de données efficace pour représenter le CCT, une organisation de mémoire adaptée à l'implémentation FPGA en utilisant la mémoire sur puce et des blocs d'accélérateur matériel pour chaque tâche.

L'architecture réalisée supporte les principales structures de données : image originale et filtrée, file d'attente de pixels, CCT, attributs et tampons de construction nécessaires pendant le calcul (l'état de l'arbre par exemple). Le système de mémoire conçu est basé sur le principe de la mémoire partagée constituée de plusieurs petits blocs de mémoire indépendants



(a) Dépendance des tâches de calcul.



(b) Architecture globale proposée par N. Ngan

FIGURE 11.6 – Architecture matérielle dédiée aux opérateurs à base du CCT.

auxquels on peut accéder en parallèle grâce au Switch Fabric. En combinant ces blocs de mémoire, nous sommes capables de réutiliser les blocs de mémoire dynamiquement.

L'accélérateur matériel de la construction de l'arbre de composantes connexes est inspirée par algorithme quasi linéaire [90], permettant de garantir que les temps d'exécution ne varient pas de manière significative avec le nombre de composants connexes.

Cependant, le temps exécution dépendra de la complexité de l'arbre, des attributs utilisés et du nombre de pixels à modifier. Compte-tenu de ces paramètres variables dans la pratique, l'application complète peut fonctionner à une cadence allant des dizaines à des centaines d'images par seconde.

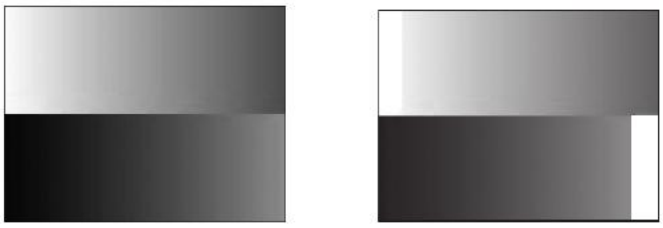
Quatre images ont été sélectionnées pour le test (Fig. 11.7) : une image en dégradé avec des transitions de niveaux de gris lisses ((a)) et trois sorties de caméra d'image infrarouge (IR) portable (b)-(d). Il s'agit des petites images ce qui est lié au contexte applicatif visé – dispositif d'affichage basse résolution pour des systèmes portables.

L'implémentation proposée [8], a été évaluée sur les images de test. Elle a permis ainsi d'obtenir une accélération jusqu'à 3 sur le Stratix II à seulement 50 MHz pour la construction de l'arbre et par rapport à une implémentation sur un PC bureau équipé avec processeur Intel Pentium IV HT (3GHz) fonctionnant sous Linux. Nous avons obtenu, par exemple, respectivement 16 ms et 21 ms pour les images de test Gradient et Man IR avec l'algorithme de Najman-Couprie sur Pentium IV Najman-Couprie [90].

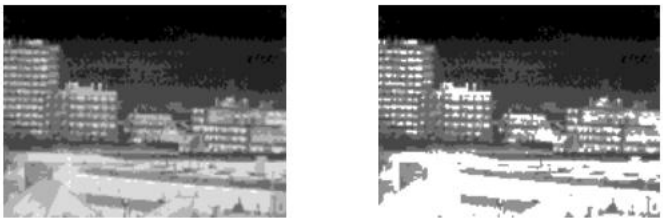
11.1.1 Implémentation parallèle sur les systèmes multi-processeur à mémoire partagée

Une deuxième approche explorée est celle par fusion des arbres 1-D a été explorée dans les travaux de thèse de P. Matas [130]. Elle résulte en un nouvel algorithme parallèle pour la construction du CCT. L'algorithme construit un arbre séparé pour chaque ligne de l'image en utilisant un algorithme 1D inspiré par [136]. Nous avons effectué une modification de l'algorithme de Menotti pour produire parent point tree [8].

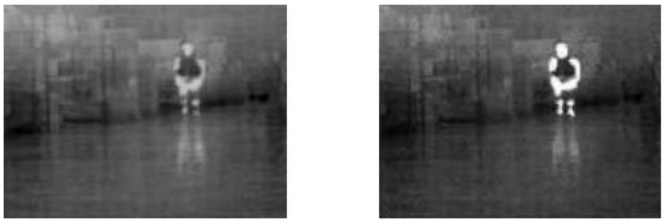
Ainsi nous construisons les arbres indépendants pour chaque ligne. Pour créer une arborescence complète de l'image composée de H lignes, les arbres de lignes adjacentes doivent



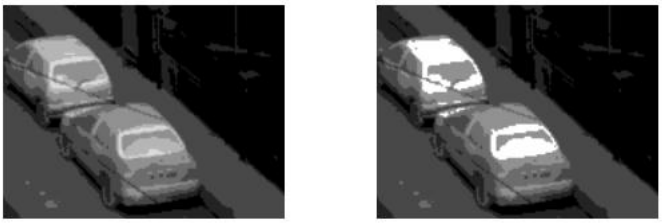
(a) Gradient.



(b) Rooftop.



(c) Man.



(d) Car.

Image	Size	Number of connected components
Man	160×120	3068
Gradient	160×120	353
Cars	160×120	84
Rooftop	160×120	362

(e) Nombre de composantes connexes présentes dans chacune des images.

FIGURE 11.7 – Images de test : l'image originale à gauche, l'image de résultat à droite.

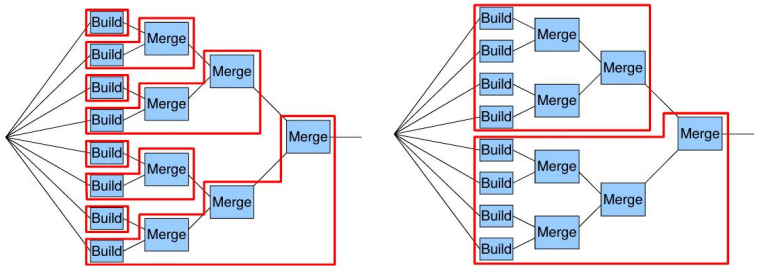
être fusionnés. Ils sont progressivement fusionnés en adaptant l'algorithme de Meijster [135] et repris dans [176].

Bien évidemment, le temps nécessaire pour la fusion est significativement plus important. que la construction des arbres 1-D. Pour cette raison, plusieurs stratégies d'ordonnancement ont été étudiées (Fig. 11.8) :

— stratégie adaptative à grain fin - maximise le degré de parallélisme de calculs

— stratégie fixe - découpe l'image d'entrée en blocs de même taille, ils sont traités par la suite en parallèle

— stratégie adaptative à gros grain - découpe l'image d'entrée en blocs dont la taille est définie par l'utilisateur, les blocs sont traités en parallèle, en fonction des ressources disponibles.



(a) Ordonnancement adaptatif à granularité fine.

(b) Ordonnancement fixe.

	Memory access optimized for	Workload balancing	Sequential input
Strategy 1	Single CPU chip	Yes	Yes
Strategy 2	Multiple CPU chips	No	No
Strategy 3	Multiple CPU chips	Yes	No

(c) Propriétés des ordonnancements évalués.

FIGURE 11.8 – Stratégies de l'ordonnancement pour la fusion des arbres 1-D.

Pour chacune des stratégies, nous avons évalué les performances : le temps d'exécution, l'équilibre de la charge des ressources de calcul ainsi que l'efficacité de calcul. Les tests ont été menés sur un cluster composé de 8 x processeurs Dual-Cores AMD Opteron à 2,6 GHz sous Suse Linux, l'image de test est de 2816x2816 en niveaux de gris sur 8bits.

Notre approche se distingue par des faibles demandes en mémoire par rapport à l'état de l'art de l'époque. Son avantage majeur est l'élimination des structures de données complexes comme des files d'attente hiérarchiques. De plus, le nouvel algorithme est intrinsèquement fortement parallèle. La parallélisation est simple, mais de bonnes stratégies d'ordonnancement ont dû être conçues pour libérer la pleine performance de l'algorithme. Les stratégies sont conçues pour occuper chaque thread matériel par un thread logiciel et le maintenir occupé le plus longtemps possible. La Fig. 11.9 permet de démontrer que les performances s'améliorent de manière linéaire avec le nombre de thread matériels disponibles.

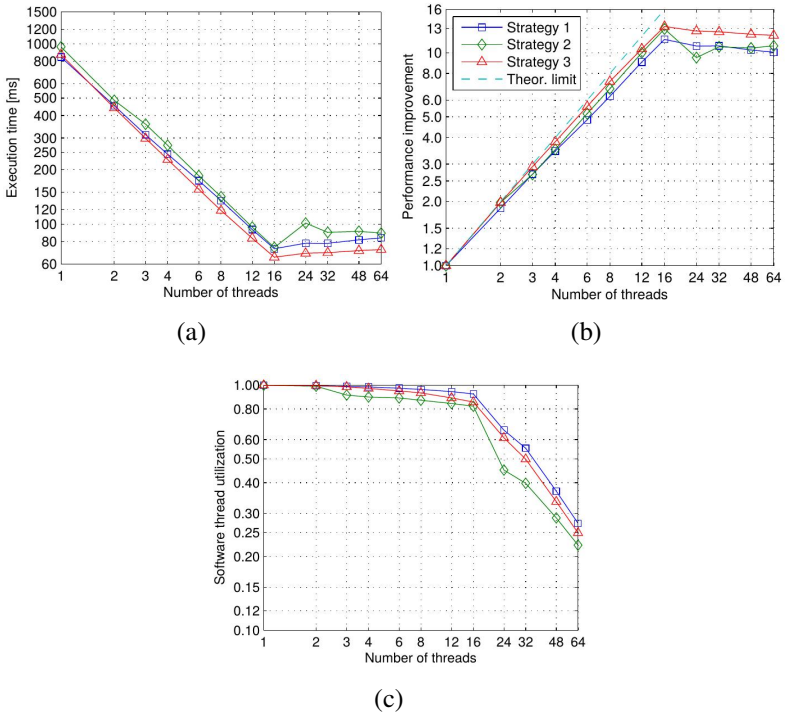
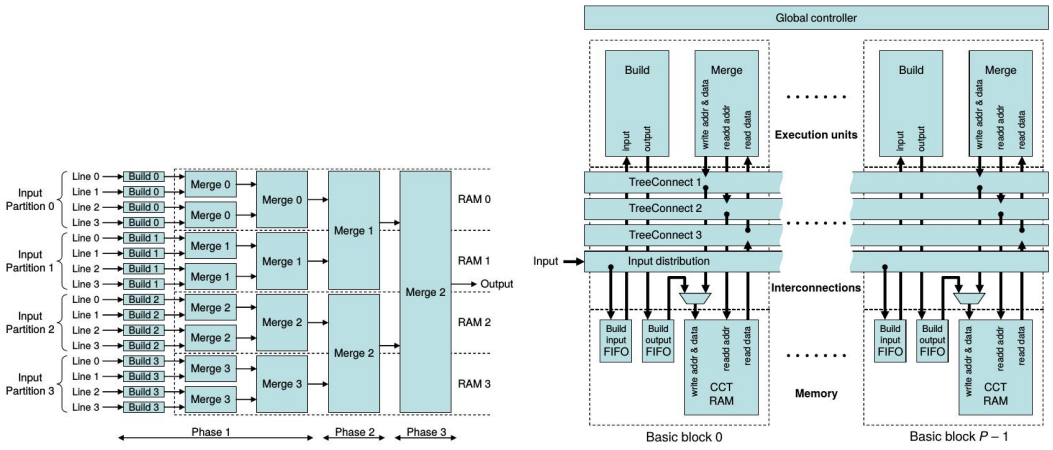


FIGURE 11.9 – Évaluation des performances obtenues avec des stratégies d'ordonnancement proposées.

11.2 Architecture matérielle dédiée aux algorithmes par fusion

Afin d'exploiter davantage le potentiel du parallélisme de l'algorithme proposé, nous nous sommes intéressés à l'exploration de l'espace des architectures dédiées.

Le flot de données que l'architecture doit supporter dans le cas idéal est illustré sur la Fig. 11.10. Il montre une partition d'image en 4 blocs pour le traitement en parallèle, chacun composé théoriquement de 4 lignes. Il montre également une association des opérations "Build" - construction de l'arbre 1D et de "Merge"- fusion, aux blocs de la RAM.



(a) Flot de données. (b) Architecture matérielle proposée.

FIGURE 11.10 – Schéma global de l'architecture matérielle de construction du CCT par fusion des arbres 1-D

L'architecture proposée est présentée sur la Fig. 11.10(b). Elle se compose d'un certain

Image size	512×512 px, 8 bits	
Image type	natural	synthetic
Total cycles	216 014	928 006
Mean cycles per pixel	0.82	3.54
Mean Merge utilization	78 %	75 %
Maximum clock frequency(a)	120 MHz	
Mean performance(b)	145 Mpx/s	34 Mpx/s
Processing time(b)	1.8 ms	7.7 ms

(a) Post-synthesis timing estimate

(b) Derived from total cycles and maximum clock frequency

(a)

	Computation resources	**Performance**
[Wilkinson 2008]	4 CPU cores @ 2.4 GHz	9.8 Mpx/s
[Matas 2008]	16 CPU cores @ 2.6 GHz	83 Mpx/s
[Ngan 2007, Ngan 2011]	Sequential HW @ 50 MHz	2.7 Mpx/s
New architecture	**16 basic blocks @ 120 MHz**	**145** Mpx/s

(b)

FIGURE 11.11 – Paramètres utilisés pour les tests et des résultats.

nombre de blocs de base, d'un réseau d'interconnexions et d'un contrôleur global. Le bloc de base est composé d'unités d'exécution (une unité de construction et une unité de fusion), de mémoires (deux FIFO et un bloc RAM CCT qui stocke l'image, l'arbre des points parents et les attributs de la partition correspondante) et d'interconnexions intrablocs.

Dans cette organisation, la scalabilité est uniquement influencée par la construction des interconnexions qui jouent un rôle important dans le processus de la fusion. L'architecture a été développée en VHDL et évaluée pour un découpage d'image en 16 blocs. Sa mise en œuvre sur un FPGA, cadencé à 120 MHz donnerait la performance de 145 Mpx sur une image de 512 × 512px. Ce qui est le résultat de 70% plus rapide que l'implémentation logicielle ([15], présentée dans la section fonctionnant sur une machine à 16 cœurs. Il est également 50 fois plus rapide que la seule architecture matérielle existante pour CCT [8] et nécessite environ 4 fois moins de mémoire.

Quatrième partie

Transfert technologique, applications

Chapitre 12

Architectures adaptables

Les dernières années marquent une tendance d'intégration de plusieurs capteurs d'images au sein des systèmes de vision portables ou mobiles. Dans des systèmes industriels, on peut désormais rencontrer des capteurs multiples, technologiquement hétérogènes : un capteur couleur, un capteur infrarouge ou un capteur basse lumière. Cette multiplication des capteurs au sein d'un seul système est la réponse aux besoins d'exploiter la complémentarité des propriétés des informations acquises liées à la sensibilité ou à la dynamique du capteur.

Bien évidemment, ces systèmes à la pointe de technologie sont souvent déployés dans des applications complexes et exigeantes sur la performance de calcul (latence et temps de traitement). Ce contexte motive le développement des nouvelles architectures matérielles émergentes dont le rôle est de proposer des solutions performantes, mais suffisamment flexibles à plusieurs niveaux : i) gestion des flux de données des différentes sources, hétérogènes en termes de résolution, de granularité de pixels et de la vitesse d'acquisition ; ii) supporter la multiplicité des cas d'utilisation et d'applications.

Dans ce contexte, j'ai pu contribuer aux deux grandes problématiques des partenaires industriels : i) définition d'un processeur reconfigurable embarqué pour les applications de la sécurité automobile, ii) calculateur embarqué adaptable pour des équipements militaires portables. La première contribution a été réalisée dans le cadre d'un projet européen MEDEA+ CarVision [82]. La deuxième problématique a donné lieu à une longue collaboration avec le Centre d'Excellence Systèmes optroniques de SAFRAN Defense. Cette collaboration a permis de réaliser deux thèses CIFRE.

12.1 Vision embarquée pour automobile

Le projet CARVISION visait à développer une plate-forme innovant dans les systèmes de vision de jour et de nuit dans le domaine de sécurité automobile. Pour cela, il impliquait la proposition de deux capteurs optiques innovants, accompagnés du système de calcul approché, pour les applications avancées d'assistance au conducteur (systèmes ADAS [1]) : un capteur CMOS à faible coût pour un capteur de caméra à haute dynamique ; un capteur infrarouge (IR [2]) basé sur la technologie du micro-bolomètre.

En tant que chercheur permanent au CEA-LIST, j'ai contribué, en tant que responsable d'une partie du projet : à la conception d'une architecture de processeur embarqué, proche des capteurs pour le traitement des applications de l'ADAS ; à l'adaptation de ce processeur pour un système d'alerte de sortie de voie et la détection des piétons.

1. ADAS - Advanced Driver-Assistance System
2. IR - Infra-Red

Les applications ciblées par le projet sont illustrées sur la Fig. 12.1 et listées sur la Fig. 12.2. Il s'agit des applications devenues aujourd'hui relativement classiques.

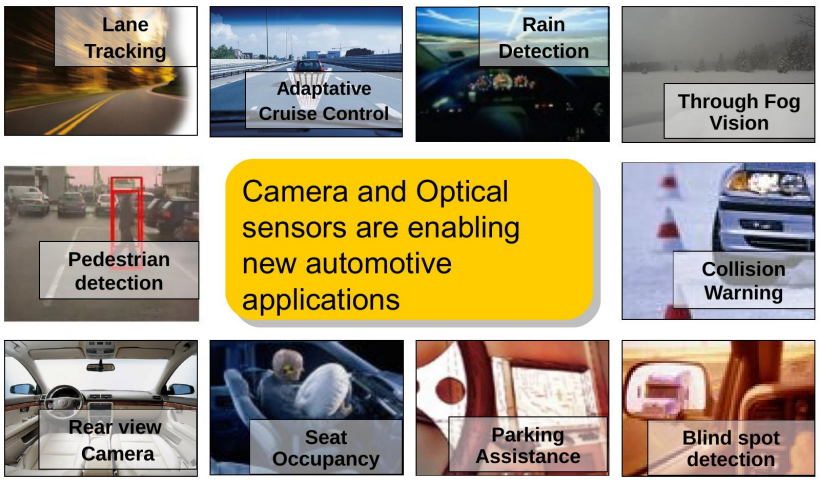


FIGURE 12.1 – Les applications considérées dans le projet CARVISION.

Function	Resolution	FPS	Dynamics (dB)	Field of view
Blind Spot Detection (BSD)	SVGA	75	140 / 170	60
Lane Change Decision Aid System (LCDAS)	MPIXEL	75	140 / 170	60
Pedestrian detection IR	CIF	20	70	30
Pedestrian detection COLOUR	VGA	15	140 / 170	45
Display Control	4 pixels	5	120	NA
Parking assistance	SVGA	30	140 / 170	60-120
Lane Departure Warning System (LDWS)	VGA	30	140 / 170	60
Light condition detection	QCIF	5	70	10
Rain detection	QCIF	5	70	10
Tunnel detection	QCIF	30	140 / 170	10
Dimming detection (condensation)	QCIF	5	70	10
Fog detection	QCIF	5	120	10
Adaptive Front Lighting System	VGA	30	120	45
Crossing vehicle	CIF	30	120	45
Night vision FAR IR	CIF	40	70	30
Night vision NEAR IR	VGA	40	140 / 170	45

FIGURE 12.2 – Les spécifications des propriétés des capteurs de vision pour les applications ADAS. La couleur orange met en évidence les applications prioritaires.

La Fig. 12.2 détaille également les spécifications de chaque application que le système devrait supporter. On peut y voir la résolution d'image et la fréquence de traitement minimales imposant les exigences sur la puissance de calcul. La dynamique du capteur a des conséquences sur la définition des pixels et la précision de calculs et besoin de mémorisation.

Pour répondre aux exigences de performance, de flexibilité et de la meilleure utilisation des ressources, nous avons proposé un système sur puce (SoC [3]) reconfigurable à gros grain avec des chemins de données programmables. La partie reconfigurable est étroitement liée à un processeur embarqué d'un côté et mémoire sur puce de l'autre. Le système est complété par un deuxième processeur dans le rôle du contrôleur principal, d'un vidéo-processeur et des interfaces pour les capteurs (Fig. 12.3).

3. SoC - System on Chip

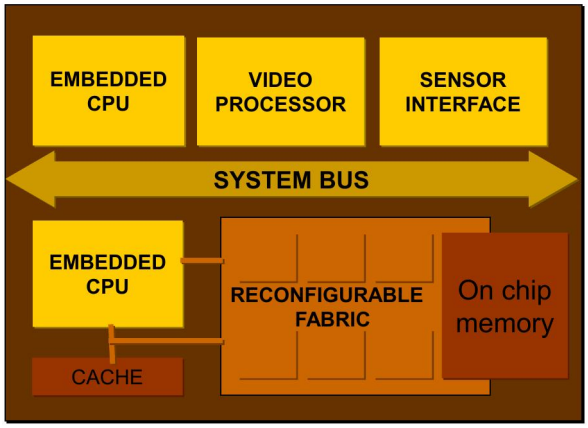
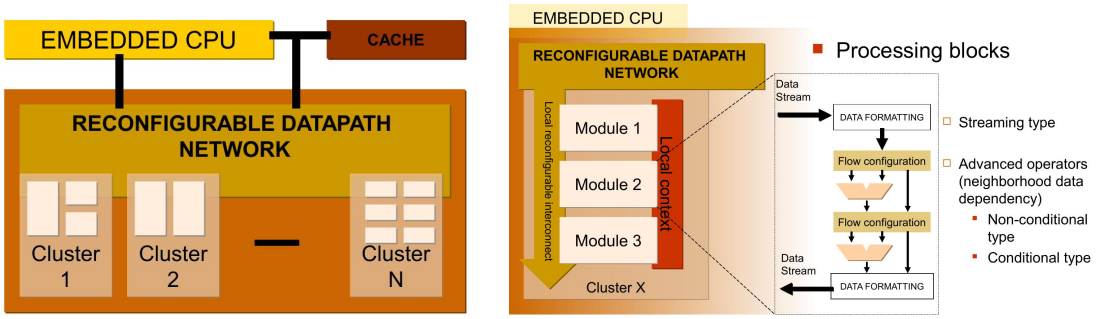


FIGURE 12.3 – Architecture globale du processeur reconfigurable proposé dans le cadre du projet CarVision.

L'architecture interne de la partie reconfigurable s'appuie sur un réseau d'interconnexions hiérarchiques et programmables permettant une gestion des flux de données et un enchaînement d'opérations flexibles. Les unités de calcul sont hautement pipelinées afin de réduire la latence de calculs (Fig. 12.4).

Les unités de calculs peuvent être hétérogènes. Leur hétérogénéité réside dans le modèle d'exécution (conditionnel et non conditionnel), degré de parallélisme et type d'accès au voisinage, pour lequel elles peuvent être spécialisées. Les unités de calcul sont réunies dans des clusters pour exécution de tâches.



(a) (b)

FIGURE 12.4 – Organisation interne du processeur reconfigurable CarVision.

Pour la conception de cette architecture, nous avons opté pour une approche "co-design" en SystemC et VHDL. Les performances ont été évaluées pour l'implémentation en VHDL, synthétisée en technologie ASIC ST Microelectronic. Nous avons utilisé l'application type de "Lane departure warning" [34], pour validation des opérateurs fondamentaux et l'organisation des ressources de calcul. Nous avons pu obtenir jusqu'à 100 fps en résolution VGA pour le calcul des contours par opérateur de Deriche, et jusqu'à 300 fps pour la détection des lignes. Également, l'étude de portage de la détection des piétons dans le flux couleur et IR a été menée. Toutes les unités de calcul nécessaires ont été implantées.

Le travail est résumé par deux brevets [56] et [57] et deux publications [33] et [34].

12.2 Systèmes de vision portables multi-capteur

Les travaux de recherche présentés dans cette section ont été effectués dans le cadre de deux thèses CIFRE [144, 116] avec le Centre d'Excellence Optronique de la société SAFRAN Defense.

FIGURE 12.5 – Exemple des équipements de vision embarquée de SAFRAN ED [116].

Nos contributions se situent dans le contexte d'évolution de l'architecture matérielle pour des systèmes de vision multi-capteurs qui sont employés dans des missions opérationnelles militaires. Au cours d'une seule mission l'environnement d'utilisateur peut changer : urbain, marin, boisé ou autre... (Fig. 12.6). Le système d'observation est ainsi exposé aux variations des conditions de luminosité par changement de l'environnement etou du jour et nuit. Des différentes fonctionnalités peuvent être choisies par utilisateur : fusion multispectrale, vision panoramique, vision multi-champs et autres.

FIGURE 12.6 – Illustration des environnements opérationnels.

Dans ce contexte, le vrai défi est de proposer un équipement portable performant avec le maximum de fonctionnalités, mais aussi apportant un support à l'utilisation ergonomique pour limiter le besoin d'interventions humaines, en cours de mission, pour régler les paramètres du système (type et nombre des capteurs utilisés, paramètres des capteurs). Cela en garantissant aux clients finaux : une qualité d'image excellente, un poids et un encombrement minimal avec une autonomie accrue du système.

Nous avons mené les recherches dans deux axes principaux :

FIGURE 12.7 – Différents modes d'utilisation [116].

- Flexibilité dynamique du système pour apporter le support matériel à l'adaptation du système aux variations de contexte opérationnel et fonctionnel permettant une meilleure utilisation des ressources de calcul.

- Prise de décision autonomes concernant les adaptations et les réglages de paramètres du système suite des changements du contexte fonctionnel et opérationnel.

12.2.1 Réseaux sur puce adaptable pour systèmes de vision multi-capteur

Pour améliorer la flexibilité du système, dans la thèse de N. Ngan [144], nous avons proposé un nouveau réseau de communications sur puce (NoC [4]) pour des systèmes de vision portable multi-capteur (Fig. 12.9).

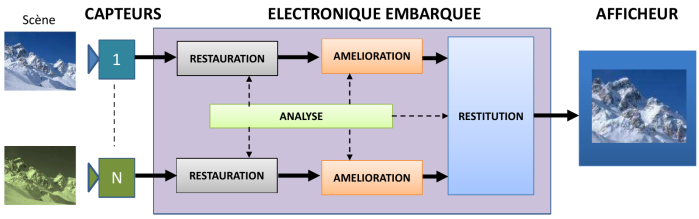


FIGURE 12.8 – Composantes standard des systèmes de vision embarqués et multi-capteur SAFRAN.

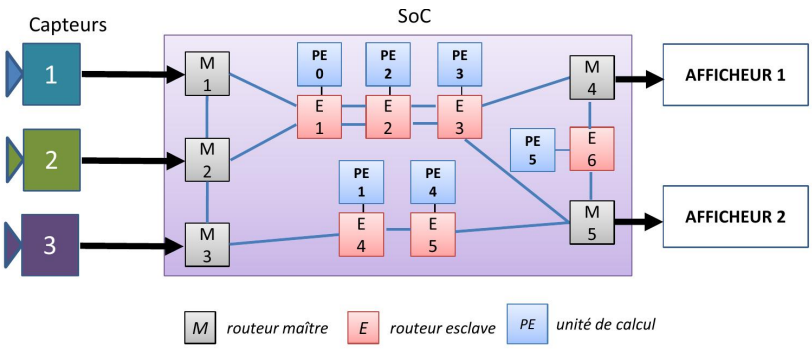


FIGURE 12.9 – Principe de mutualisation des unités de calcul à l'aide d'un réseaux sur puce.

Ce réseau est construit à partir de deux types de routeurs : le routeur maître et le routeur esclave. Les nœuds maîtres sont connectés de façon indirecte par des nœuds esclaves. D'un point de vue système, il s'agit d'un réseau indirect de routeurs maîtres qui sont reliés par plusieurs réseaux linéaires directs de routeurs esclaves (Fig. 12.10).

Un nœud maître représente une source ou une destination principale pour un paquet de données. En particulier, un routeur maître est caractérisé par des interfaces avec le monde extérieur lui permettant d'acquérir un flux de données entrant et de sortir un flux de données traité. L'architecture des routeurs a été publiée en [40].

4. NoC - Network on Chip

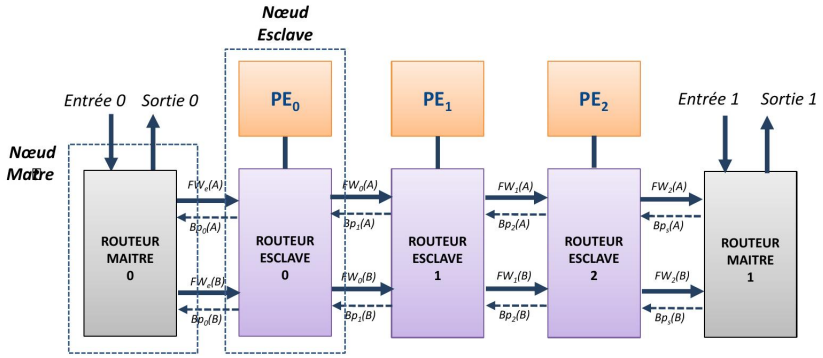


FIGURE 12.10 – Bloc de base réseau linéaire des routeurs esclaves entre les routeurs maîtres.

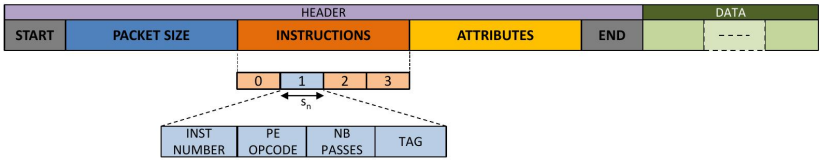


FIGURE 12.11 – Format des paquets contenant des instructions sur le traitement des flux.

Dans ce réseau, nous avons intégré une méthode originale de communication des commandes dédiées aux routeurs esclaves. Elle propose insérer des commandes (instructions) concernant le traitement des données directement dans le flux de données.

Pour cela, une nouvelle organisation du paquet de données est définie, complétée par une nouvelle méthode d'aiguillage, appelée Split-Wormhole switching, qui est adaptée à notre routeur esclave (Fig. 12.11). L'algorithme de routage dynamique associée à ce routeur permet de décider de la modification du chemin en fonction de l'adresse de destination et des instructions contenues dans l'en-tête du flux de données [41].

Dans le cas des systèmes multi-capteurs, la gestion de plusieurs flux pose la question de la gestion des accès à la mémoire. Nous complétons donc notre approche par une proposition d'un partitionnement de la mémoire spécifique à trames-images partagée [43].

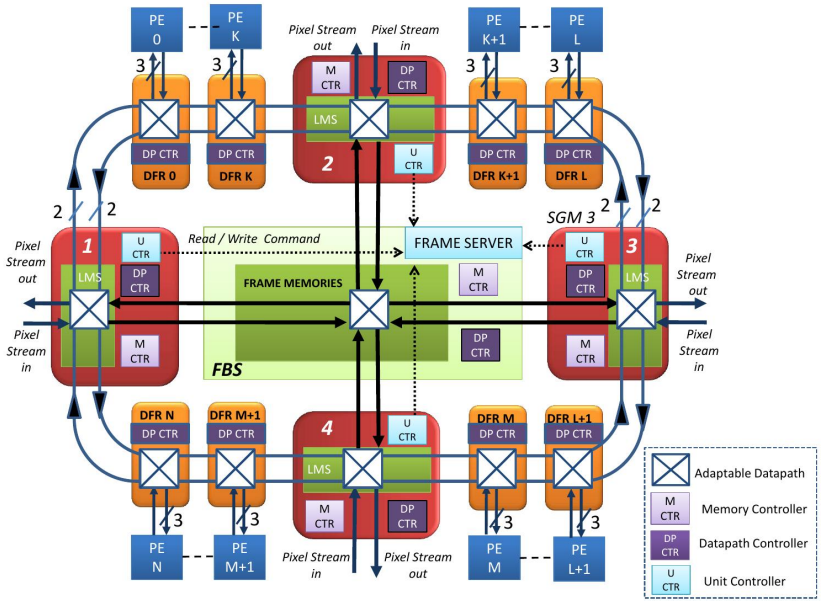


FIGURE 12.12 – Multi Data Flow Ring : architecture dotée d'un NOC adaptable.

Pour son évaluation, nous organisons ce réseau de communication sur puce en anneau

appelé Multi Data Flow Ring (Fig. 12.12), exploitant les différentes unités de routage et le système de mémoire de trames partagée. Son fonctionnement a été validé dans le cadre d'une application choisie de la visualisation d'image, typiquement utilisée dans des équipements de vision embarquée de SAFRAN Defense. Un prototype FPGA a été proposé afin d'évaluer les latences d'adaptations des chemins de données dans le réseau, les latences de chargement des contextes ainsi que la surface consommée.

Des résultats obtenus ont montré que les temps d'adaptation restent pertinents. L'impacte des latences d'adaptation est amorti avec la taille croissante du paquet de données.

Nous avons obtenu un surcoût de 10 % de surface sur un composant FPGA de grande matrice (150k éléments logiques), qui reste acceptable par rapport à la taille des unités de calcul utilisée. Ce surcoût est largement compensé par les possibilités d'adaptation dynamique d'une application en cours de fonctionnement.

12.2.2 Moniteur pour système auto-adaptable

Les travaux précédents ont contribué à lever les verrous de flexibilité mais ne fournissent pas la solution de prise de décision et de contrôle de l'ensemble. C'est le rôle du moniteur-système qui devrait permettre à l'architecture actuelle de SAFRAN Defense de s'auto-adapter dynamiquement. Cela a été la cible des travaux d'une deuxième thèse CIFRE, réalisée par d'Ali Isavudeen.

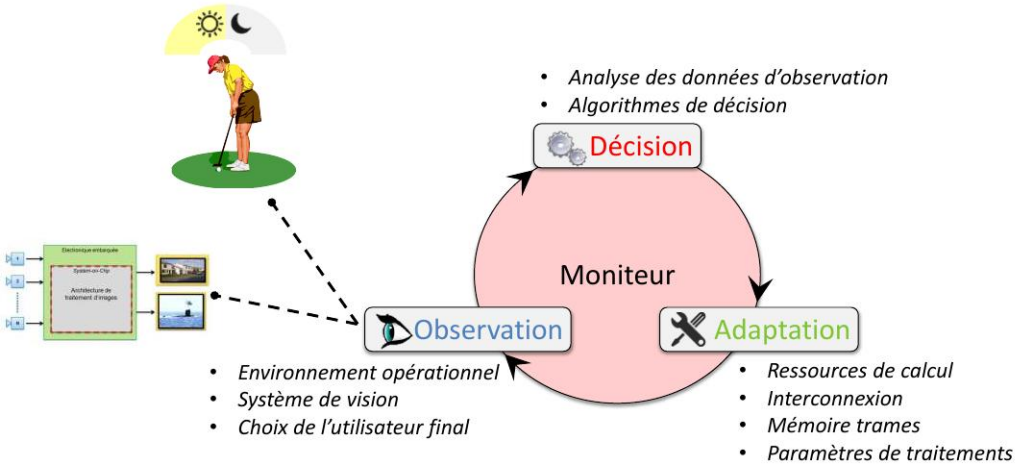


FIGURE 12.13 – Rôle du moniteur d'un système adaptable.

L'idée de base est qu'un moniteur de l'architecture observe les changements du contexte d'utilisateur et de l'état du système. Puis, il décide et il pilote les adaptations d'architecture nécessaires à effectuer pour réaliser par exemple : le changement de capteur, l'ajout d'un capteur, la modification de fréquence de traitement, le changement d'application (Fig. 12.13).

Dans la thèse d'Ali Isavudeen, nous étudions un réseau sur puce dont le principal objectif est l'acheminement des données d'observation vers le moniteur et distribuer les commandes d'adaptation vers les différentes parties de celle-ci. Le défi majeur est d'accéder aux informations sur l'état de l'architecture sans pour autant compromettre ses performances.

Pour délivrer les performances exigées par le contexte opérationnel (Fig. 12.14), le moniteur proposé prend en charge la gestion automatique d'adaptation des paramètres ou de l'organisation des ressources de calculs et des chemins des données. Ainsi l'architecture peut s'auto-adapter dynamiquement aux variations de contexte.

Les adaptations sont matérialisées par des commandes que le moniteur envoie aux différents contrôleurs de l'architecture. Nous avons exploré plusieurs mécanismes d'adaptation :

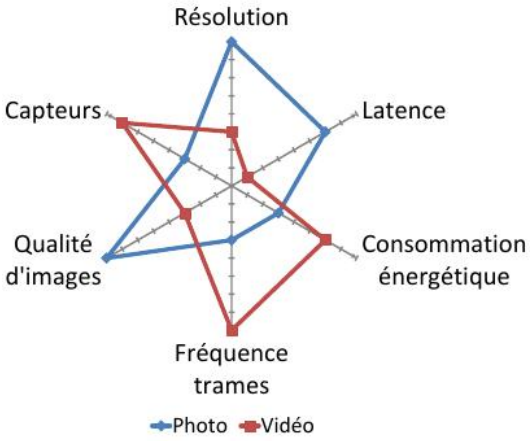


FIGURE 12.14 – Évaluation des exigences des différents mode opérationnel.

i) reconfiguration partielle et dynamique pour disposer des ressources de calculs adaptés à la tâche ;

ii) adaptation de la fréquence d'une chaîne de traitement par reconfiguration de PLL [5] ; cette adaptation voit son intérêt dans le cas d'un changement portant sur la résolution et/ou la fréquence trames du flux d'images

iii) adaptation de la mémoire des trames ; un changement des valeurs de résolution et/ou de fréquence trames peut exiger une adaptation de l'espace mémoire.

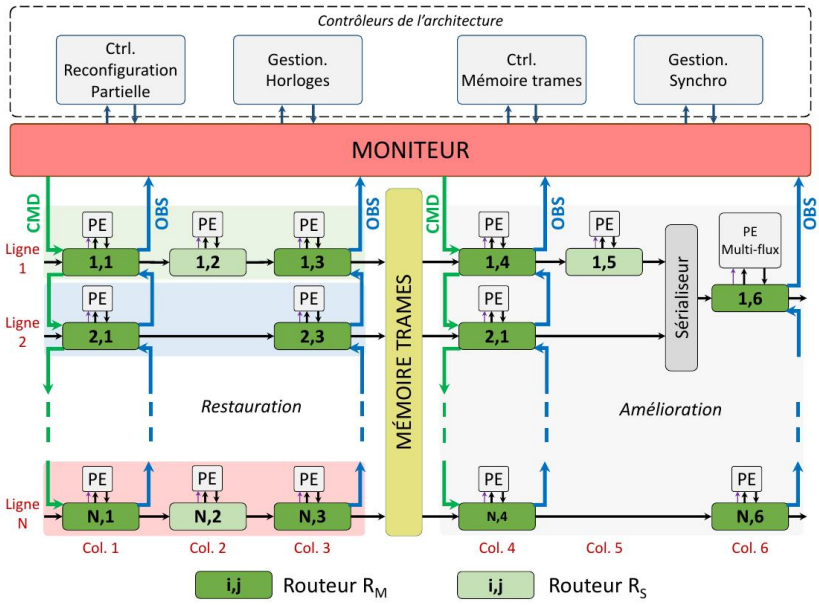


FIGURE 12.15 – Réseaux sur puce proposé pour observation et commande d'un système auto-adaptable.

Nous exploitons la proposition de la thèse de N. Ngan [144] des commandes intégrés directement dans le flux des données. Nous introduisons deux types de routeurs (Fig. 12.15) : RM - Routeurs du Monitoring et les RS - Routeurs Simples. Un routeur RM est un routeur d'interface entre le moniteur et le réseau de routeurs tandis qu'un routeur RS se situe à l'intérieur du réseau. Nous utilisons donc un système de paquets de données muni d'un en-tête contenant les données d'observation. Cet en-tête est également utilisé pour les données

5. PLL - Phase-Locked Loop

de l'arbitrage lors du routage des paquets de données. Quel que soit le nombre de capteurs, donc de chaînes de traitement, ce réseau relie le moniteur à l'architecture avec seulement un nombre limité des liaisons.

Nous avons implémenté cette solution sur une cible matérielle de type FPGA. Nous avons évalué ses performances en surface par une synthèse sur un FPGA Cyclone V de chez ALTERA (5CGX). La solution proposée présente un faible surcoût en surface par rapport au bilan surfacique d'une architecture multi-flux de référence. Les latences d'auto-adaptation obtenues respectent les contraintes de temps dont nous disposons. Les étapes d'observation et de décision ne représentent que quelques dizaines cycles d'horloges. Notons toutefois que les latences des adaptations matérielles par reconfiguration dynamique partielle dépendent directement de la cible d'implémentation. Ces résultats sont détaillés dans les publications suivantes : [14, 36, 46, 37] et sont résumé dans le manuscrit de thèse [116].

Chapitre 13

Applications de vision embarquée

13.1 Détection et identification des cibles pour drones (UAV)

Aujourd'hui, nous assistons à une exploration massive des champs d'application des drones (UAVs). Leur utilisation est prévue non seulement dans des applications classiques d'inspection, de surveillance, mais également, à plus grande échelle dans la construction, gestion de crise et bien sure dans les transports et les livraisons.

À relativement court horizon de temps, il est prévu que ces robots volants évoluent de manière autonome et collaborent avec d'autres robots ou même des humains. La conscience de l'environnement devient alors une capacité indispensable pour ces engins. Dans ce contexte, la vision embarquée temps-réel a un rôle prépondérant.

Dans ce contexte, nous abordons le problème de la compréhension de la scène dans un environnement extérieur ou intérieur non-contrôlé. Le défi est de savoir comment gérer avec des images dans des conditions impactées par des ombres, changement d'échelle, perspective, vibration, bruit, flou, entre autres.

À travers des travaux de la thèse d'Eric Bazan (Mines-ParisTech) que je co-encadre, nous essayons traiter le problème mentionné en imitant ce processus de la perception humaine.

Les humains peuvent mener le processus de perception de manière naturelle [149]. Les primitives sont identifiées en tant qu'une conséquence de leur apparition non-accidentelle, c'est-à-dire qu'ils ne sont pas générés aléatoirement [68, 97].

Nous avons proposé une procédure de détection et de reconnaissance d'une cible d'atterrissage, basée sur un modèle de perception (principe de Helmholtz et la théorie de la Gestalt). Celle-ci est illustrée sur la Fig. 13.1.

L'approche été validée sur des images des cibles d'atterrissage avec des déformations simulées et réelles. Nous avons testé l'algorithme dans une base de données d'images synthétiques qui simule quatre dégradations d'image : bruit, ombres, déformation de la cible et changement de taille (Fig. 13.2). Pour les situations réelles, nous avons effectué plusieurs tests dans des scénarios intérieurs et extérieurs (https ://youtu.be/igsQc7VEF2c).

Les résultats ont fait l'objet de deux publications [45, 10]. Ces travaux ont été réalisés en réponse à un problème industriel et les résultats s'insèrent dans mes activités liées aux projets DiXite et UrbaRiskLab.

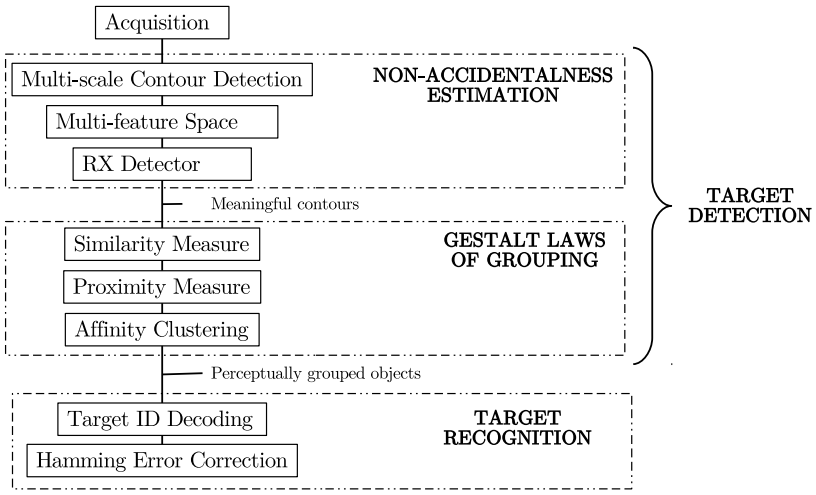


FIGURE 13.1 – Différentes phases de la détection et de la reconnaissance d'une cible selon les principes de perception visuelle.

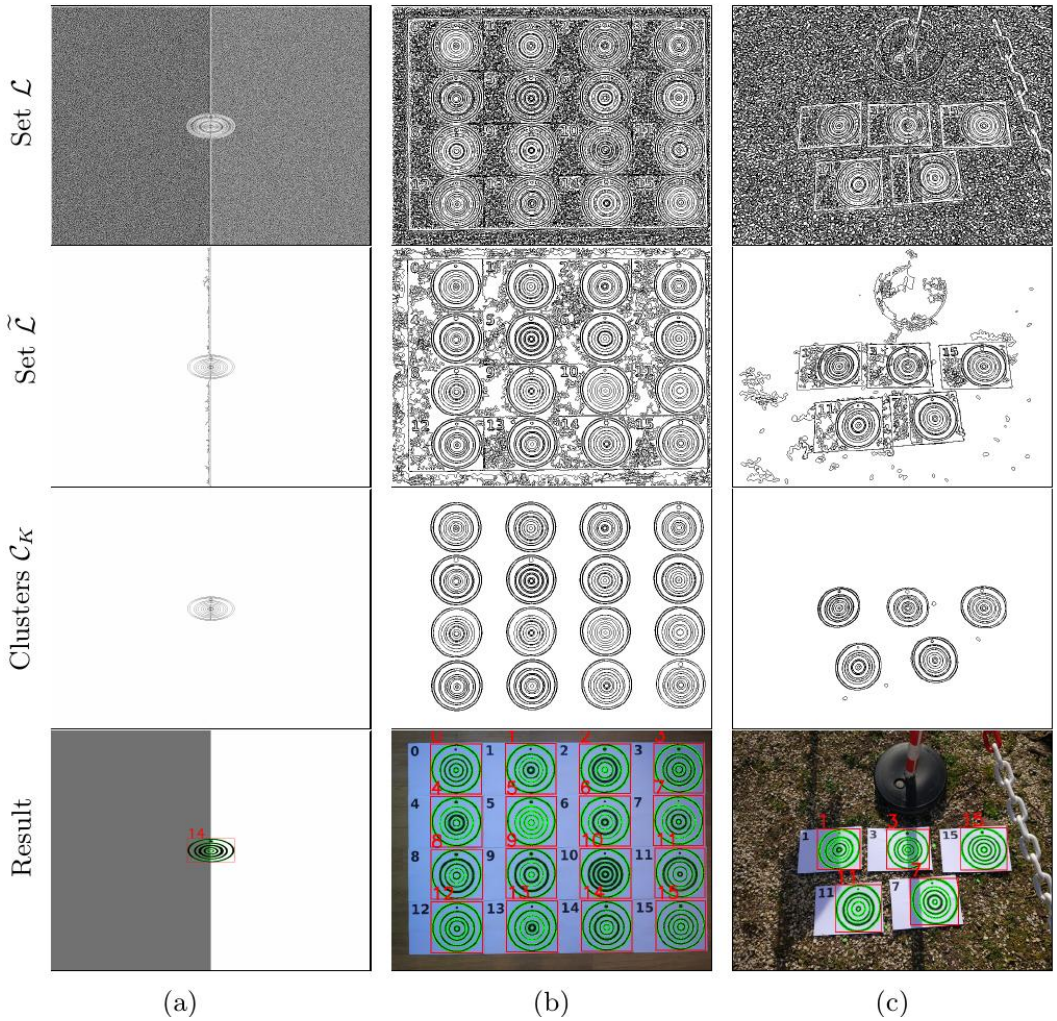


FIGURE 13.2 – Détection de cible basée sur les principes de perception : conditions simulées.

13.2 Classification des particules détectées par Timepix

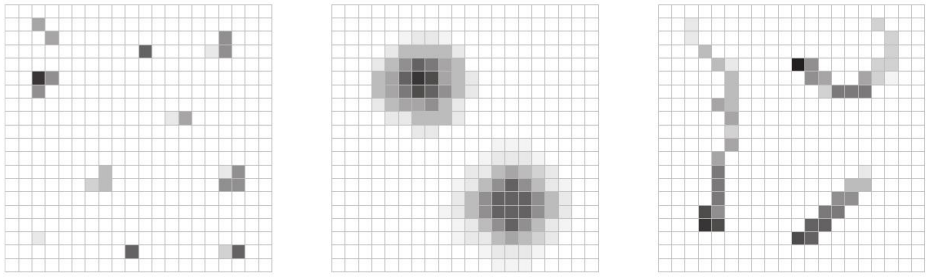
Le capteur Timepix est une nouvelle génération de détecteurs CMOS, dérivés du détecteur Medipix2 et conçus par le CERN - permettant trois types de mesures : i) "temps

d'arrivée" (ToA), ii) "dépassement du seuil", iii) comptage [128].

Les trois modes de fonctionnement permettent une large gamme d'applications ; des observatoires astronomiques, imagerie par rayons X et de nombreuses expériences physiques (analyse des flux radioactifs des sources radioactives inconnues). Le détecteur permet d'utiliser une vitesse d'échantillonnage élevée et ainsi d'enregistrer jusqu'à 3000 images par seconde (fps).

Un grand intérêt est porté à l'utilisation de ces détecteurs pour la reconstruction des événements physiques. Cela consiste en reconnaissance des particules chargées et leur identification (par exemple. :α , γ , électrons), suivie de quelques mesures statistiques (c'est-à-dire les particules majeures et leur angle d'incidence) [75]. La reconnaissance des particules nécessite d'identifier et d'analyser et de caractériser les formes des traces, représentant la "signature" de la particule, laissée chaque fois qu'une particule "touche" le détecteur Timepix (Fig. 13.3).

Le problème majeur est de proposer des méthodes de classification permettant d'atteindre la fréquence de trame élevée et ainsi d'acquérir des images avec une fréquence d'échantillonnage élevée, réduisant ainsi le risque d'apparition des particules qui se chevauchent et qui ne peuvent pas être classées.



(a) dots (b) blobs (c) tracks

FIGURE 13.3 – Forme des différentes particules : dots = γ, blobs = α, tracks = *lctrons*

Dans [20] nous avons proposé une méthode de classification efficace des traces de particules en utilisant les marqueurs morphologiques complétés par des filtres non-linéaires au lieu de descripteurs basés sur des composants connectés, qui nécessiteraient beaucoup de calcul (Fig. 13.4).

Nous avons proposé l'architecture matérielle dédiée sur un FPGA. Le traitement peut être réalisé en flots de données sans mémorisation intermédiaire réduisant la latence de calcul. Ceci nous a permis d'atteindre de très hautes performances, allant jusqu'à plus de 700 images par seconde.

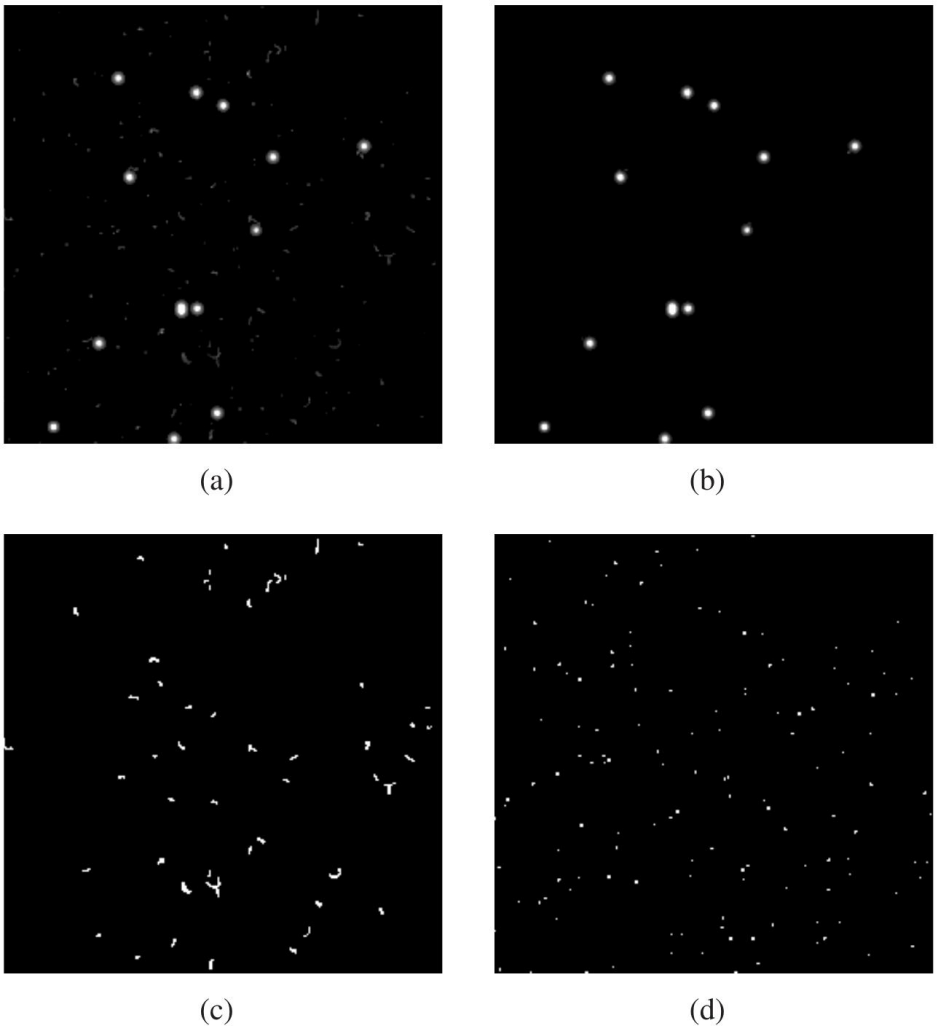


FIGURE 13.4 – Exemple des résultats obtenus : images contenant à des particules séparées selon leur type.

Cinquième partie

Projet de recherche, perspectives

Chapitre 14

Projet de recherche

Avant de dresser un tableau de mon projet de recherche, je propose de rappeler mes axes passés dans la continuité desquels mes orientations futures s'inscrivent. Il sera aussi utile d'évoquer quelques réflexions concernant le contexte industriel et sociétal qui permettra de mieux comprendre ces orientations.

Contributions théoriques passées

Mes contributions scientifiques, présentées dans le manuscrit, peuvent être également vues de manière suivante :

— **Morphologie mathématique continue** - calcul massivement parallèle et asynchrone de la fonction distance, avec une précision sous-pixellique (2D et 3D) pour segmentation d'images et de séquences vidéo ;

— **Morphologie mathématique discrète** - algorithme de calcul des opérateurs concaténés en temps constant, avec mémoire et latence de calcul optimale pour des filtres de grande tailles et angles arbitraires ;

— **Morphologie mathématique et graphes** - algorithme efficace et parallèle pour la construction des arbres des composantes connexes.

À chaque domaine est liée au moins une innovation technologique, c'est-à-dire une nouvelle réalisation matérielle efficace ou une réalisation parallèle sur des processeurs multi-cœurs ou GPU. Chacune de ces propositions atteignait des performances dépassant l'état de l'art à la date de la publication :

— calcul globalement asynchrone de l'algorithme de ligne de partage des eaux (LPE) permettant de simplifier radicalement la gestion d'accès aux données tout en améliorant le temps d'exécution,

— réalisation matérielle dédiée pour le calcul des opérateurs de traitement d'image formulés à l'aide de l'arbre de composantes connexes (min-max tree). Une telle réalisation a été considérée comme impossible auparavant.

Certaines de ces contributions ont permis des innovations industrielles :

— Détection, reconnaissance et classification des particules pour dosimètre intelligent avec le capteur TimePix (IPEA, Prague)

— Conception et validation d'un processeur embarqué reconfigurable pour la sécurité automobile (systèmes ADAS) (FIAT, FICOSA, ST-Microelectronics).

— Architecture matérielle auto-adaptable pour des systèmes de vision multi-capteurs (SAFRAN-Defense).

14.1 Nouveau contexte de recherche

Depuis les dernières années, nous assistons au déploiement intensif des capteurs de vision, et ne pouvons que constater l'accélération et l'intensification de besoin d'un traitement d'images automatisé, dans de nombreux domaines : installations urbaines, usine du futur, chantier du futur, contrôle non-destructif des procédés, équipements agricoles, dans des voitures, des drones, etc., sans parler des téléphones portables et tablettes. Par exemple, il est estimé que les installations urbaines seules représenteront une base installée de presque un milliard de capteurs d'ici à 2020 (Source : Nvidia).

Dans ce contexte, la tendance est que les méthodes de traitement d'image s'approchent de plus en plus des capteurs pour des raisons de mobilité, de réactivité locale ou pour des limitations des besoins de transmission.

En parallèle, cette tendance s'accompagne d'une explosion des volumes et des variétés de données définies par la technologie des capteurs (couleur, profondeur, infra-rouge,...) et donc à traiter : définition des pixels hétérogènes, les résolutions au-delà de la HD, dimensions supérieures à 3D pour certains capteurs spécifiques.

Pour cela, des nouvelles approches émergent. Elles bénéficient d'avancement des méthodes d'apprentissage et de perception. En revanche, il me semble indispensable de tenir compte des contraintes réelles dès la conception de ces méthodes de traitement d'images. Notamment celles proches des capteurs ou embarquées sur des systèmes mobiles qui doivent, par exemple, être adaptées aux données hétérogènes, exhiber un fort potentiel de parallélisation ou faire appel à des modèles réduits.

Pour tenir compte de ce contexte, je souhaiterais à l'avenir de développer un thème de recherche que j'intitule **Nouvelles méthodes de perception et de la compréhension de scène, proches de capteur** et que je présente dans les paragraphes suivants, à l'horizon de court et moyen terme.

Ce thème comporte plusieurs composantes, certaines représentant la suite directe des travaux de recherche précédents, certaines sont nouvelles mais complémentaires.

14.2 Projet de recherche à court et moyen terme

Extension de MM [1] vers les graphes aux poids négatifs

Si par le passé le watershed (Ligne de Partage des Eaux) a été considéré comme l'outil de segmentation d'image par excellence, il apparaît toujours d'actualité soit - avec les marqueurs : comme un outil de segmentation interactive, soit - non contraint : comme un outil de pré-traitement ou de simplification d'image. Cette tendance est attestée par nombre de publications récentes dans ce domaine [119, 91, 138, 137, 122].

Son importance même s'accroît depuis qu'il a trouvé sa nouvelle application dans le domaine de masses de données (big data) [139, 98].

Dans ce contexte et dans le prolongement de Massive Marching, que j'ai introduit en 2003, il est possible d'ouvrir de nouvelles perspectives. Parmi elles on peut citer en exemple des récents travaux sur des calculs parallèles des distances pondérées sur des graphes [83, 123, 127]. Ainsi, un axe intéressant d'extension de Massive Marching serait de proposer :

— la généralisation du Massive Marching pour le calcul sur des graphes valués aux poids négatifs, en combinant le schéma numérique existant avec un schéma de propagation modifié, inspiré de l'algorithme de Bellman-Ford [118, 70]

1. MM - algorithme Massive Marching

— un nouvel formalisme sous forme de notation matricielle avec un calcul réalisé sur des matrices creuses. Cela ouvrirait la porte à la parallélisation des calculs dont l'importance est résumée dans l'étude récente de Hong [113] et de Weber [174].

Les avantages sont de deux natures i) éliminer la nécessité d'appliquer le "swamping" lorsque la fonction distance est utilisée pour le calcul du watershed avec marqueurs, ii) réaliser le calcul en utilisant des librairies de calcul matriciel avec des matrices creuses disponibles aujourd'hui sur des plate-formes à base de GPU.

Segmentation et analyse d'image en utilisant des principes de perception visuelle

Si l'axe précédent s'inscrit dans la continuation des travaux antérieurs, l'axe présenté ici représente l'exploration d'une nouvelle approche à la compréhension de la scène. Les principes de perception du cortex visuel commencent a être formulé mathématiquement [108, 0].

De manière générale, il s'agit d'étudier les diverses combinaisons de distributions d'intensité, de patterns texturaux locaux ou de formes géométriques en distributions multimodales et procéder par agrégation de régions sur la base de distances de distributions. Par exemple, le max-tree [156], tree of shapes [64] ou peak-decomposition [65] proposent une décomposition d'image simple et rapide à calculer [39]. D'un autre côté une segmentation probabiliste de textures a été proposée [117] permettant d'agréger des régions sur la base de la distribution de l'intensité.

Une autre approche est représentée par les modèles a contrario [96] dédiés à la détection des déviations statistiquement significatives. Ils permettent de proposer des techniques avancées de segmentation ou d'extraction d'objets perceptuellement saillants.

À travers la thèse d'Eric Bazan (Mines-ParisTech), ces recherches ont été déjà initiées, les premiers résultats semblent prometteurs. Nous avons pu proposer un modèle perceptuel non supervisé, basé sur des contours, pour la détections et reconnaissance des zones d'atterrissages des drones [10].

Comme phase suivante, il serait intéressant d'étendre notre démarche à des régions, en combinant justement les propriétés et les mesures statistiques à base de texture, couleur, d'intensité.

14.3 Évolution à plus long terme

À plus long terme, je souhaiterais orienter le projet de recherche énoncé vers le champ applicatif de la compréhension de la scène pour la navigation des véhicules autonomes terrestres ou aériens. Avec l'arrivée de l'automatisation de la mobilité, la compréhension de scène pour localisation et navigation deviennent un problème fondamental de recherche. Ce problème comprend deux parties : i) navigation à base de la vision et ii) la modélisation de l'environnement. Pour cela, une interaction est nécessaire avec les domaines suivants : reconstruction 3D, analyse de textures, segmentation des images et reconnaissance des formes, estimation de la profondeur, utilisation des images multi-spectrales, mais aussi des algorithmes rapides et parallèles.

Par conséquent, la recherche dans cet axe propose un potentiel fort de synergie avec des axes développés au sein des équipes des laboratoires regroupés au sein de l'UPE et du LIGM. Les axes proposés sont également en phase avec les défis adressés dans les projets « tremplin » I-SITE DiXite et UrbaRiskLab.s

Méthodes de perception pour UAV [2]

A moyen terme, ce projet de recherche vise à développer des outils pour modélisation de la scène en temps réel, dans des conditions non-contrôlées. Le modèle en 2D ou 3D de la scène serait enrichi par une labellisation sémantique des éléments importants, permettant de représenter et localiser (en 2-D ou 3-D) à la fois les structures dans l'environnement : sol, murs, ciel et les éléments importants de ces structures : ouvertures (fenêtres, portes), escaliers. Ainsi progressivement, avec le déplacement de la caméra, le modèle enrichi, par exemple représenté par une carte labellisée, serait créé et utilisé pour l'aide à la navigation automatisée, qui n'est pas très précise actuellement du fait des pertes de signal GPS en ville ou à l'intérieur.

Idéalement, je souhaiterais considérer à l'entrée des séquences vidéo sans contrainte, avec une vitesse de déplacement qui n'est connue qu'approximativement (à l'aide d'autres capteurs) et peut varier brutalement.

Un des défis majeurs est de proposer une méthodologie pleinement automatique, robuste aux changements de luminosité et la moins dépendante d'un modèle a priori.

Le travail à mener peut se décomposer en plusieurs parties :

- Modélisation de la scène - ici, je souhaiterais étendre mes compétences et étudier les approches monoculaires, temps-réel, pour construction rapide, robuste et précise du modèle 3D, compatible avec la fusion des informations de la segmentation sémantique [154, 169]. Il est à noter que les méthodes actuelles sont très gourmandes en mémoire et nécessitant des calculs coûteux sur GPU.

- Segmentation et compréhension de la scène - le travail peut s'inspirer des propositions évoquées dans la partie Recherche à court terme. De plus, des approches morphologiques générales proposent un ensemble complet d'outils de segmentation et d'analyse de texture nécessaire et l'identification des régions de la scène, il serait intéressant de travailler sur leur adaptation au contexte énoncé.

Également, ce travail pourrait viser les méthodes de segmentation hiérarchique, pouvant être combinées avec les réseaux convolutionnels ce qui est actuellement une tendance très forte [101].

- Apprentissage invariant sous rotation - la classification des images par apprentissage profond a dépassé la précision de l'état de l'art des tâches confiées à l'intelligence artificielle. Le problème est que ceci se fait au détriment de la complexité du modèle et du besoin de grands volumes de données : grandes bases de données et beaucoup de paramètres.

Lorsque nous considérons les conditions non contrôlées, l'invariance aux symétries (rotations, changement d'échelle, symétrie) est requise. Mais cela complexifie encore plus le modèle. L'équivariance en rotation est généralement résolue par l'augmentation des données. Cela améliore certainement la généralisation, mais n'est pas exact, ne parvient pas à saisir l'équivariance locale, et n'assure pas l'équivariance de chaque couche d'un réseau. En outre, le plus gros inconvénient est la taille accrue du classificateur. Certaines expériences ont été également réalisées avec Harmonic Network [177] ou Scattering Network [79]. Une autre possibilité consiste à utiliser un espace de rotation, obtenu en utilisant un filtre orienté, pivoté dans toutes les directions et en empilant le résultat. À travers la thèse de Rosemberg Rodriguez, nous avons initié les premières explorations de cette approche. Les premiers résultats semblent encourageants et ils ont été communiqués lors du Collège doctoral franco-allemand en novembre 2018.

2. Unmanned Aerial Vehicles

14.4 Rayonnement scientifique

Par le passé, j'ai eu l'occasion de contribuer au rayonnement de l'ESIEE Paris par développement de mon initiative des relations internationales : double diplôme, convention des échanges ERASMUS avec plusieurs universités étrangères, ou encore très récemment, représentation de l'école doctorale MSTIC à la tournée des études doctorales au Mexique.

Dans l'avenir, je souhaiterais m'investir dans l'organisation des événements scientifiques tels que des réunions GDR, de colloques, d'écoles-d'été.

Il est aussi de mon souhait de pouvoir m'impliquer davantage dans l'organisation de la vie scientifique dans des sociétés savantes, activité éditoriale (journaux, collections) ou d'évaluateur (projets ANR, clusters européens).

Chapitre 15

Projet d'enseignement

Mes activités d'enseignement, mon expérience des différentes approches pédagogiques et la proximité avec plusieurs domaines de la recherche me permettent de contribuer à plusieurs niveaux de formation : cycle ingénieur, formation par la recherche (en master et en cursus doctoral), formation par apprentissage ou formation continue.

J'ai une expérience riche en création des nouveaux modules d'enseignement, soit de mon initiative pour compléter les compétences dans un parcours des élèves de filière Informatique ; sois à la demande de l'institution pour compléter les compétences fondamentales d'un programme. J'ai également l'habitude de dispenser mes cours devant un public international (cours en anglais).

Depuis plusieurs années, j'ai acquis l'expérience d'enseigner et d'adapter les méthodes pédagogiques aux promotions aux compétences hétérogènes, telles que les filières par apprentissage ou programmes-passerelles.

Pour illustrer, sur le plan des approches pédagogiques, j'apprécie les enseignements guidés par la pratique, tels que la classe inversée, le coaching pédagogique ou l'apprentissage par projet. Il me semble important de donner des exemples pratiques permettant aux étudiants de mieux visualiser les solutions aux divers problèmes (conceptuels ou pratiques). Cela permet d'une part de faciliter la compréhension des outils théoriques nécessaires ainsi que d'autre part améliorer l'implication et la motivation des élèves.

Mes domaines d'intervention sont variés (traitement d'image, systèmes de vision embarquée, programmation, réalisations parallèles) et me permettent de proposer des enseignements pluridisciplinaires ou fondamentaux.

À l'avenir, je souhaiterais pouvoir créer un cours du niveau Master **Perception pour navigation des véhicules autonomes** : de la spécification à la réalisation et son application à la navigation des véhicules autonomes. La vision, avec des liens avec l'intelligence artificielle, va produire l'information utile en entrée du système de contrôle ou d'asservissement d'un véhicule autonome.

References

[64] MONASSE, P. AND GUICHARD, F. "Fast computation of a contrast invariant image representation". In : *IEEE Transactions on Image Processing* 9.5 (2000), 860–872.

[65] R. ALAIS, P. DOKLÁDAL, E. DECENCIÈRE et B. FIGLIUZZI. "Function Decomposition in Main and Lesser Peaks". In : *ISMM 2017*. 2017.

[66] C. V. ALVINO, G.B. UNAL, G.G. SLABAUGH, B. PENY et T. FANG. "Efficient segmentation based on Eikonal and diffusion equations". In : *International Journal of Computer Mathematics* 84.9 (2007), p. 1309 -1324. URL : http://openaccess.city.ac.uk/4410/.

[67] G. ANELLI, A. BROGGI et G. DESTRI. "Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms". In : *IEEE Trans. Pattern Anal. Mach. Intell.* 20.2 (1998), p. 217-224.

[68] Fred ATTNEAVE. "Some Informational Aspects of Visual Perception." In : *Psychological Review* 61.3 (1954), p. 183-193. ISSN : 1939-1471(Electronic),0033-295X(Print). DOI : 10.1037/h0054663.

[69] Jan BARTOVSKY. "Hardware architectures for morphological filters with large structuring elements". Theses. Université Paris-Est, nov. 2012. URL : https://tel.archives-ouvertes.fr/tel-00788984.

[70] R. BELLMAN. "On a Routing Problem". In : *Quarterly of Applied Mathematics* 16 (1958), p. 87-90.

[71] C. BERGER, T. GERAUD, R. LEVILLAIN, N. WIDYNSKI, A. BAILLARD et E. BERTIN. "Effective Component Tree Computation with Application to Pattern Recognition in Astronomical Imaging". In : *2007 IEEE International Conference on Image Processing*. T. 4. 2007, p. 41-44.

[72] Harry BLUM. "Biological shape and visual science (part I)". In : *Journal of Theoretical Biology* 38.2 (1973), p. 205 -287. ISSN : 0022-5193.

[73] R. van den BOOMGAARD et D. A. WESTER. "Logarithmic shape decomposition". In : *Aspects of Visual Form Processing*. Sous la dir. de C. ARCELLI, L. P. CORDELLA et G. Sanniti di BAJA. Capri, Italy. World Scientific Publishing Co., Singapore, 1994, p. 552-561.

[74] P. BOSILJ, T. DUCKETT et G. CIELNIAK. "Connected attribute morphology for unified vegetation segmentation and classification in precision agriculture". In : *Computers in Industry* 98 (2018), p. 226 -240. ISSN : 0166-3615.

[75] J. BOUCHAMI, A. GUTIÉRREZ, T. HOLY, A. HOUDAYER, J. JAKŮBEK, C. LEBEL, C. LEROY, J. MACANA, J.-P. MARTIN, S. POSPÍŠIL, S. PRAK, P. SABELLA et C. TEYSSIER. "Measurement of pattern recognition efficiency of tracks generated by ionizing radiation in a Medipix2 device". In : *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 633 (2011). 11th International Workshop on Radiation Imaging Detectors (IWORID), S187 -S189. ISSN : 0168-9002. DOI : https://doi.org/10.1016/j.nima.2010.06.163. URL : http://www.sciencedirect.com/science/article/pii/S0168900210013537.

[76] M. BOUÉ et P. DUPUIS. "Markov Chain Approximations for Deterministic Control Problems with Affine Dynamics and Quadratic Cost in the Control". In : *SIAM Journal on Numerical Analysis* 36.3 (1999), p. 667-695.

[77] J. E. BRESENHAM. "Algorithm for computer control of a digital plotter". In : *IBM Systems Journal* 4.1 (1965), p. 25-30. ISSN : 0018-8670. DOI : 10.1147/sj.41.0025.

[78] R. W. BROCKETT et P. MARAGOS. "Evolution equations for continuous-scale morphology". In : *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*. T. 3. 1992, p. 125-128.

[79] Joan BRUNA et Stéphane MALLAT. "Invariant Scattering Convolution Networks". In : *CoRR* abs/1203.1513 (2012). arXiv : 1203.1513. URL : http://arxiv.org/abs/1203.1513.

[80] A. CAPOZZOLI, C. CURCIO, A. LISENO et S. SAVARESE. "A comparison of Fast Marching, Fast Sweeping and Fast Iterative Methods for the solution of the eikonal equation". In : *2013 21st Telecommunications Forum Telfor (TELFOR)*. 2013, p. 685-688.

[81] Joshua E. CATES, Aaron E. LEFOHN et Ross T. WHITAKER. "GIST: an interactive, GPU-based level set segmentation tool for 3D medical images". In : *Medical Image Analysis* 8.3 (2004). Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003, p. 217 -231. ISSN : 1361-8415.

[82] CEA-LIST. http://www.catrene.org/web/downloads/profiles_medea/2A401_profile.pdf. 2006.

[83] Theodore CHABARDES, Petr DOKLÁDAL, Matthieu FAESSEL et Michel BILODEAU. "A parallel, O(n), algorithm for unbiased, thin watershed ". In : *IEEE International Conference on Image Processing*. Phoenix, United States, sept. 2016.

[84] B. B. CHAUDHURI. "An efficient algorithm for running window pel gray level ranking 2-D images". In : *Pattern Recogn. Lett.* 11.2 (1990), p. 77-80. ISSN : 0167-8655. DOI : http://dx.doi.org/10.1016/0167-8655(90)90116-J.

[85] Y.-J. CHIANG, T. LENZ, X. LU et G. ROTE. "Simple and optimal output-sensitive construction of contour trees using monotone paths". In : *Computational Geometry* 30.2 (2005). Special Issue on the 19th European Workshop on Computational Geometry, p. 165 -195. ISSN : 0925-7721.

[86] Shao-Yi CHIEN, Shyh-Yih MA et Liang-Gee CHEN. "Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements". In : *IEEE Transactions on Circuits and Systems for Video Technology* 15.9 (2005), p. 1156-1169. ISSN : 1051-8215. DOI : 10.1109/TCSVT.2005.852622.

[87] C. CLIENTI, S. BEUCHER et M. BILODEAU. “A system on chip dedicated to pipeline neighborhood processing for Mathematical Morphology”. In : *2008 16th European Signal Processing Conference*. 2008, p. 1-5.

[88] Ch. CLIENTI, M. BILODEAU et S. BEUCHER. “An efficient hardware arhcitecture without line memories for morphological image processing”. In : *ACIVS*. 2008.

[89] D. COLTUC et I. PITAS. “On fast running max-min filtering”. In : *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on* 44.8 (1997), p. 660-663.

[90] M. COUPRIE, L. NAJMAN et G. BERTRAND. “Quasi-Linear Algorithms for the Topological Watershed”. In : *Journal of Mathematical Imaging and Vision* 22.2 (2005), p. 231-249. ISSN : 1573-7683.

[91] J. COUSTY, G. BERTRAND, L. NAJMAN et M. COURPIE. “Watershed cuts: Minimum spanning forests and the drop of water principle”. In : *Pattern Analysis and Machine Intelligence, IEEE Transactions* 31.8 (2009), p. 1362-1374.

[92] O. CUISENAIRE, E. ROMERO, C. VERAART et B. M. M. MACQ. “Automatic segmentation and measurement of axons in microscopic images”. In : t. 3661. 1999.

[93] Per Erik DANIELSSON. “Euclidean Distance Mapping”. In : *Computer Graphics and Image Processing* 14.3 (nov. 1980), p. 227-248. ISSN : 0146-664X.

[94] Olivier DÉFORGES, Nicolas NORMAND et Marie BABEL. “Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture”. In : *Journal of Real-Time Image Processing* 8.2 (2013), p. 143-152. ISSN : 1861-8219. DOI : 10.1007/s11554-010-0171-8. URL : https://doi.org/10.1007/s11554-010-0171-8.

[95] Claire-Hélène DEMARTY. “Segmentation et structuration d’un document vidéo pour la caractérisation et l’indexation de son contenu sémantique”. Thèse de doct. École Nationale Supérieure des Mines de Paris, jan. 2000.

[96] Agnès DESOLNEUX. “When the a Contrario Approach Becomes Generative”. In : *Int. J. Comput. Vision* 116.1 (jan. 2016), p. 46-65. ISSN : 0920-5691. DOI : 10.1007/s11263-015-0825-x. URL : http://dx.doi.org/10.1007/s11263-015-0825-x.

[97] Agnès DESOLNEUX, Lionel MOISAN et Jean-Michel MOREL. *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. en. Interdisciplinary Applied Mathematics. New York : Springer-Verlag, 2008. ISBN : 978-0-387-72635-9.

[98] Fabio DIAS, Moussa R MANSOUR, Paola R VALDIVIA, Jean COUSTY et Laurent NAJMAN. “Watersheds on Hypergraphs for Data Clustering”. In : sous la dir. de J. ANGULO, S. VELASCO-FORERO et F. MEYER. T. 10225. Lecture Note In Computer Sciences. Fontainebleau, France : Springer, mai 2017, p. 211-221.

[99] M. Van DROOGENBROECK et M. J. BUCKLEY. “Morphological Erosions and Openings: Fast Algorithms Based on Anchors”. In : *J. Math. Imaging Vis.* 22.2-3 (2005), p. 121-142. ISSN : 0924-9907. DOI : http://dx.doi.org/10.1007/s10851-005-4886-2.

[100] M. Van DROOGENBROECK et H. TALBOT. “Fast computation of morphological operations with arbitrary structuring elements”. In : *Pattern Recognition Letters* 17.14 (1996), p. 1451-1460. URL : citeseer.ist.psu.edu/vandroogenbroeck96fast.html.

[101] Clément FARABET, Camille COUPRIE, Laurent NAJMAN et Yann LECUN. "Learning Hierarchical Features for Scene Labeling". In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (août 2013), p. 1915 -1929. DOI : 10.1109/TPAMI.2012.231. URL : https://hal-upec-upem.archives-ouvertes.fr/hal-00742077.

[102] Michael L. FREDMAN et Robert Endre TARJAN. "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". In : *J. ACM* 34.3 (juil. 1987), p. 596-615. ISSN : 0004-5411.

[103] David Z. GEVORKIAN, Jaakko T. ASTOLA et Samvel M. ATOURIAN. "Improving Gil-Werman Algorithm for Running Min and Max Filters". In : *IEEE Trans. Pattern Anal. Mach. Intell.* 19.5 (1997), p. 526-529. ISSN : 0162-8828. DOI : http://dx.doi.org/10.1109/34.589214.

[104] Leonardo GIGLI, Santiago VELASCO-FORERO et Beatriz MARCOTEGUI. "On Minimum Spanning Tree Streaming for Image Analysis". In : *2018 25th IEEE International Conference on Image Processing (ICIP)*. Athens, Greece, oct. 2018. URL : https://hal.archives-ouvertes.fr/hal-01986210.

[105] T. GIJBELS, P. SIX, L. Van GOOL, F. CATTHOOR, H. De MAN et A. OOSTERLINCK. "A VLSI-architecture for parallel non-linear diffusion with applications in vision". In : *Proceedings of 1994 IEEE Workshop on VLSI Signal Processing*. 1994, p. 398-407.

[106] J. GIL et M. WERMAN. "Computing 2-D Min, Median, and Max Filters". In : *IEEE Trans. Pattern Anal. Mach. Intell.* 15.5 (1993), p. 504-507. ISSN : 0162-8828. DOI : http://dx.doi.org/10.1109/34.211471.

[107] Joseph (Yossi) GIL et Ron KIMMEL. "Efficient Dilation, Erosion, Opening, and Closing Algorithms". In : *IEEE Trans. Pattern Anal. Mach. Intell.* 24.12 (2002), p. 1606-1617. ISSN : 0162-8828. DOI : http://dx.doi.org/10.1109/TPAMI.2002.1114852.

[108] Rafael Grompone von GIOI, Jérémie JAKUBOWICZ, Jean-Michel MOREL et Gregory RANDALL. "LSD: a Line Segment Detector". In : *Image Processing On Line* 2 (2012), p. 35-55. DOI : 10.5201/ipol.2012.gjmr-lsd.

[109] R. GOLDENBERG, R. KIMMEL, E. RIVLIN et M. RUDZSKY. "Fast geodesic active contours". In : *IEEE Transactions on Image Processing* 10.10 (2001), p. 1467-1475. ISSN : 1057-7149. DOI : 10.1109/83.951533.

[110] H. HEDBERG, P. DOKLADAL et V. OWALL. "Binary Morphology With Spatially Variant Structuring Elements: Algorithm and Architecture". In : *IEEE Transactions on Image Processing* 18.3 (2009), p. 562-572. ISSN : 1057-7149.

[111] M. van HERK. "A Fast Algorithm for Local Minimum and Maximum Filters on Rectangular and Octagonal Kernels". In : *Pattern Recognition Letters* 13.7 (1992), p. 517-521.

[112] Sverker HOLMGREN et Dan WALLIN. "Performance of High-Accuracy PDE Solvers on a Self-Optimizing NUMA Architecture". In : *Euro-Par 2001 Parallel Processing*. Sous la dir. de Rizos SAKELLARIOU, John GURD, Len FREEMAN et John KEANE. Berlin, Heidelberg : Springer Berlin Heidelberg, 2001, p. 602-610. ISBN : 978-3-540-44681-1.

[113] Sumin HONG et Won-Ki JEONG. "A Multi-GPU Fast Iterative Method for Eikonal Equations Using on-the-fly Adaptive Domain Decomposition". In : *Procedia Computer Science* 80 (2016). International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, p. 190 -200. ISSN : 1877-0509. DOI : https://doi.org/10.1016/j.procs.2016.05.309. URL : http://www.sciencedirect.com/science/article/pii/S1877050916306676.

[114] T.S. HUANG, G.J. YANG et G.Y. TANG. "A Fast Two-Dimensional Median Filtering Algorithm". In : *IEEE Trans. Acoustics, Speech and Signal Processing* 27.1 (1979), p. 13-18.

[115] K. HWANG, P. S. TSENG et D. KIM. "An orthogonal multiprocessor for parallel scientific computations". In : *IEEE Transactions on Computers* 38.1 (1989), p. 47-61. ISSN : 0018-9340.

[116] Ali ISAVUDEEN. "Architecture Dynamiquement Auto-adaptable pour Systèmes de Vision Embarquée Multi-capteurs". Thèse de doctorat dirigée par Akil, Mohamed et Dokladalova, Eva Informatique Paris Est 2017. Thèse de doct. 2017. URL : http://www.theses.fr/2017PESC1071.

[117] Dominique JEULIN. "Morphological probabilistic hierarchies for texture segmentation". In : *Mathematical Morphology - Theory and Applications* 1.Issue 1 (2016), p. 216-234.

[118] Lester R. Ford JR. "Network Flow Theory". In : *RAND Corporation*. Santa Monica, California, 1956.

[119] C. KAUFFMAN et N. PICHE. "Cellular automaton for ultra-fast watershed transform on GPU". In : *19th International Conference on Pattern Recognition, ICPR* (2008).

[120] S. KIM et D. FOLIE. "The group marching method: An level set eikonal solver". In : *SEG Technical Program Expanded Abstracts 2000*. 2000, p. 2297-2300.

[121] R. KIMMEL, N. KIRYATI et A. M. BRUCKSTEIN. "Sub-pixel distance maps and weighted distance transforms". In : *Journal of Mathematical Imaging and Vision* 6.2 (1996), p. 223-233. ISSN : 1573-7683.

[122] J. KOLOMAZNIK. "Interactive Processing of Volumetric Data". Thèse de doct. Faculty of mathematics et physics, CVUT Prague, 2017.

[123] Jan KOLOMAZNÍK, Jan HORACEK, Václav KRAJÍCEK et Josef PELIKÁN. "Implementing Interactive 3D Segmentation on CUDA Using Graph-Cuts and Watershed Transformation". In : *Proc. 20th WSCG Int. Conf. Computer Graphics, Visualization and Computer Vision, 2012, pp. 1–4*. 2012.

[124] Daniel LEMIRE. "Streaming Maximum-Minimum Filter Using No More than Three Comparisons per Element". In : *CoRR* (2006).

[125] Daniel LEMIRE. "Streaming maximum-minimum filter using no more than three comparisons per element". In : *Nordic J. of Computing* 13.4 (2006), p. 328-339. ISSN : 1236-6064.

[126] F. LEMONNIER et J.-C. KLEIN. "Fast Dilation by large 1D Structuring Elements". In : *IEEE International Workshop on Nonlinear Signal and Image Processing, Halkidiki, Greece*. 1995.

[127] J. LINDBLAD et N. SLADOJE. "Exact Linear Time Euclidean Distance Transforms of Grid Line Sampled Shapes". In : *Mathematical Morphology and Its Applications to Signal and Image Processing*. Sous la dir. de J. A. BENEDIKTSSON et al. Cham : Springer International Publishing, 2015, p. 645-656. ISBN : 978-3-319-18720-4.

[128] X. LLOPART, R. BALLABRIGA, M. CAMPBELL, L. TLUSTOS et W. WONG. "Timepix, a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements". In : *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 581.1 (2007). VCI 2007, p. 485 -494. ISSN : 0168-9002. DOI : https://doi.org/10.1016/j.nima.2007.08.079. URL : http://www.sciencedirect.com/science/article/pii/S0168900207017020.

[129] R. MALLADI et J. A. SETHIAN. "Level set and fast marching methods in image processing and computer vision". In : *Proceedings of 3rd IEEE International Conference on Image Processing*. T. 1. 1996, 489-492 vol.1. DOI : 10.1109/ICIP.1996.559540.

[130] Petr MATAS. *Connected component tree construction for embedded systems*. Thèse de l'Université Paris-Est. Juin 2014.

[131] G. MATHERON. *Random Sets and Integral Geometry*. New York : John Wiley et Sons, 1975.

[132] Georges MATHERON. *Eléments pour une théorie des milieux poreux*. Paris : Masson, 1967.

[133] J. MATTES et J. DEMONGEOT. "Efficient Algorithms to Implement the Confinement Tree". In : *Discrete Geometry for Computer Imagery*. Sous la dir. de G. BORGEFORS et al. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000, p. 392-405. ISBN : 978-3-540-44438-1.

[134] J. MATTES et J. DEMONGEOT. "Efficient Algorithms to Implement the Confinement Tree". In : *Discrete Geometry for Computer Imagery*. Sous la dir. de G. BORGEFORS, I. NYSTRÖM et G.S. di BAJA. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000, p. 392-405. ISBN : 978-3-540-44438-1.

[135] A. MEIJSTER. "Efficient sequential and parallel algorithms for morphological image processing". English. University of Groningen. Thèse de doct. 2004. ISBN : 903671978X.

[136] D. MENOTTI-GOMES, L. NAJMAN et A. DE ALBUQUERQUE ARAUJO. "1D Component tree in linear time and space and its application to gray-level image multithresholding". In : *International Symposium on Mathematical Morphology*. Sous la dir. de Banon Gerald Jean FRANCIS et al. T. 1. 1. France : INPE, 2007, p. 437-448.

[137] F. MEYER. "A watershed algorithm progressively unveiling its optimality". In : *Proc. Mathematical Morphology and its Applications to Signal and Image Processing* (2015), p. 717-728.

[138] F. MEYER. "The steepest watershed: from graphs to images". In : *Computer Vision and Pattern Recognition* abs/1024.2134 (2012).

[139] S. V. MITSYN et G. A. OSOSKOV. "Watershed on vector quantization for clustering of big data". In : *Physics of Particles and Nuclei Letters* 12.1 (2015), p. 170-172. ISSN : 1531-8567. DOI : 10.1134/S1547477115010173. URL : https://doi.org/10.1134/S1547477115010173.

[140] T. MIYATAKE, M. EJIRI et H. MATSUSHIMA. “A fast algorithm for maximum-minimum image filtering”. In : *Systems and Computers in Japan* 27.13 (1996), p. 74-85.

[141] V. MORARD, P. DOKLADAL et E. DECENCIERE. “Linear openings in arbitrary orientation in O (1) per pixel”. In : *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE. 2011, p. 1457-1460.

[142] Laurent NAJMAN. *Skew detection*. European Patent. Filled at August 27, 2002 as a European filing the French Patent Office. 2002.

[143] Laurent NAJMAN et Michel SCHMITT. “Watershed of a continuous function”. In : *Signal Processing* 38.1 (1994). Mathematical Morphology and its Applications to Signal Processing, p. 99 -112. ISSN : 0165-1684.

[144] Nicolas NGAN. “Etude et conception d’un réseau sur puce dynamiquement adaptable pour la vision embarquée”. Thèse de doctorat dirigée par Akil, Mohamed Informatique Paris Est 2011. Thèse de doct. 2011. URL : http://www.theses.fr/2011PEST1040.

[145] Bogusł aw OBARA. “Identification of transcrystalline microcracks observed in microscope images of a dolomite structure using image analysis methods based on linear structuring element processing”. In : *Comput. Geosci.* 33.2 (2007), p. 151-158. ISSN : 0098-3004.

[146] *Open Source Computer Vision Library*. URL : https://opencv.org.

[147] Stanley OSHER et James A SETHIAN. “Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations”. In : *Journal of Computational Physics* 79.1 (1988), p. 12 -49. ISSN : 0021-9991.

[148] J. PECHT. “Speeding up successive Minkowski operations”. In : *Pattern Recognition Letters* 3.2 (1985), p. 113-117.

[149] Jean PETITOT. *Neurogéométrie de la vision: modèles mathématiques et physiques des architectures fonctionnelles*. fr. Editions Ecole Polytechnique, 2008. ISBN : 978-2-7302-1507-7.

[150] P. PISCAGLIA, A. CAVALLARO, M. BONNET et D. DOUXCHAMPS. “High Level Description of Video Surveillance Sequences”. In : *Multimedia Applications, Services and Techniques — ECMAST’99*. Sous la dir. d’Helmut LEOPOLD et Narciso GARCÍA. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999, p. 316-331. ISBN : 978-3-540-48757-9.

[151] Frederic PRECIOSO et Michel BARLAUD. “B-Spline Active Contour with Handling of Topology Changes for Fast Video Segmentation”. In : *EURASIP Journal on Advances in Signal Processing* 2002.6 (2002), p. 793184. ISSN : 1687-6180.

[152] Thomas RETORNAZ. “Automatic detection of text from natural scenes: a semantic descriptor for content based image retrieval”. Theses. École Nationale Supérieure des Mines de Paris, oct. 2007.

[153] M. RUMPF et R. STRZODKA. “Nonlinear Diffusion in Graphics Hardware”. In : *Data Visualization 2001*. Sous la dir. de David S. EBERT, Jean M. FAVRE et Ronald PEIKERT. Vienna : Springer Vienna, 2001, p. 75-84. ISBN : 978-3-7091-6215-6.

[154] Yohann SALAÜN, Renaud MARLET et Pascal MONASSE. "Line-based Robust SfM with Little Image Overlap". In : *3DV 2017 International Conference on 3D Vision 2017*. Qingdao, China, oct. 2017. URL : https://hal-enpc.archives-ouvertes.fr/hal-01618686.

[155] P. SALEMBIER, A. OLIVERAS et L. GARRIDO. "Antiextensive connected operators for image and sequence processing". In : *IEEE Transactions on Image Processing* 7.4 (1998), p. 555-570. ISSN : 1057-7149.

[156] Philippe SALEMBIER, Albert OLIVERAS, et Luis GARRIDO. "Antiextensive connected operators for image and sequence processing". In : *Image Processing, IEEE Transactions on* 7.4 (1998), 555–570.

[157] J. SERRA. *Image Analysis and Mathematical Morphology*. T. 2. Academic Press, NY, 1988, p. 40-46.

[158] J. A. SETHIAN. "Fast Marching Methods". In : *SIAM Review* 41.2 (1999), p. 199-235.

[159] J.A. SETHIAN. "Evolution, Implementation, and Application of Level Set and Fast Marching Methods for Advancing Fronts". In : *Journal of Computational Physics* 169.2 (2001), p. 503 -555. ISSN : 0021-9991. DOI : https://doi.org/10.1006/jcph.2000.6657. URL : http://www.sciencedirect.com/science/article/pii/S0021999100966579.

[160] J.A. SETHIAN. *Level set methods : Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science*. T. 3. Cambridge Monographs on Applied and Computational Mathematics. Demandeur: F. Meyer No Inventaire 48/98. Cambridge University Press, 1996.

[161] K. SIDDIQI, B. B. KIMIA et Chi-Wang SHU. "Geometric shock-capturing ENO schemes for subpixel interpolation, computation, and curve evolution". In : *Proceedings of International Symposium on Computer Vision - ISCV*. 1995, p. 437-442.

[162] C. SIGG, R. PEIKERT et M. GROSS. "Signed distance transform using graphics hardware". In : *IEEE Visualization, 2003. VIS 2003.* 2003, p. 83-90.

[163] Peter SMEREKA. "Semi-Implicit Level Set Methods for Curvature and Surface Diffusion Motion". In : *J. Sci. Comput.* 19 (2003), p. 439-456.

[164] Pierre SOILLE. *Morphological Image Analysis: Principles and Applications*. 2e éd. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2003. ISBN : 3540429883.

[165] Pierre SOILLE, Edmond J. BREEN et Ronald JONES. "Recursive Implementation of Erosions and Dilations Along Discrete Lines at Arbitrary Angles". In : *IEEE Trans. Pattern Anal. Mach. Intell.* 18.5 (1996), p. 562-567. ISSN : 0162-8828. DOI : http://dx.doi.org/10.1109/34.494646.

[166] M. SONKA, V. HLAVAC et R. BOYLE. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007. ISBN : 049508252X.

[167] Vinh-Thong TA, Abderrahim ELMOATAZ et Olivier LÉZORAY. "Adaptation of Eikonal Equation over Weighted Graph". In : *Scale Space and Variational Methods in Computer Vision*. Sous la dir. de Xue-Cheng TAI, Knut MØRKEN, Marius LYSAKER et Knut-Andreas LIE. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 187-199. ISBN : 978-3-642-02256-2.

[168] Robert TARJAN. "Efficiency of a Good But Not Linear Set Union Algorithm". In : 22 (avr. 1975), p. 215-225.

[169] Marvin TEICHMANN, Michael WEBER, J. Marius ZÖLLNER, Roberto CIPOLLA et Raquel URTASUN. "MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving". In : *CoRR* 1612.07695 (2016). arXiv : 1612.07695. URL : http://arxiv.org/abs/1612.07695.

[170] Y. TSAI. *Rapid and Accurate Computation of the Distance Function Using Grids*. Rapp. tech. 17. University of California, USA, 2000.

[171] E. R. URBACH et M. H. F. WILKINSON. "Efficient 2-D Grayscale Morphological Transformations With Arbitrary Flat Structuring Elements". In : *IEEE TIP* 17.1 (2008), p. 1-8.

[172] P.W. VERBEEK et B.J.H. VERWER. "Shading from shape, the eikonal equation solved by grey-weighted distance transform". In : *Pattern Recognition Letters* 11.10 (1990), p. 681 -690. ISSN : 0167-8655.

[173] Ofir WEBER, Yohai S. DEVIR, Alexander M. BRONSTEIN, Michael M. BRONSTEIN et Ron KIMMEL. "Parallel Algorithms for Approximation of Distance Maps on Parametric Surfaces". In : *ACM Trans. Graph.* 27.4 (nov. 2008), 104:1-104:16. ISSN : 0730-0301.

[174] Ofir WEBER, Yohai S. DEVIR, Alexander M. BRONSTEIN, Michael M. BRONSTEIN et Ron KIMMEL. "Parallel algorithms for approximation of distance maps on parametric surfaces". In : *ACM Trans. Graph.* 27 (2008), 104:1-104:16.

[175] J. WEICKERT, B. M. T. H. ROMENY et M. A. VIERGEVER. "Efficient and reliable schemes for nonlinear diffusion filtering". In : *IEEE Transactions on Image Processing* 7.3 (1998), p. 398-410. ISSN : 1057-7149.

[176] M. H. F. WILKINSON, H. GAO, W. H. HESSELINK, J. E. JONKER et A. MEIJSTER. "Concurrent Computation of Attribute Filters on Shared Memory Parallel Machines". In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.10 (2008), p. 1800-1813. ISSN : 0162-8828.

[177] Daniel E. WORRALL, Stephan J. GARBIN, Daniyar TURMUKHAMBETOV et Gabriel J. BROSTOW. "Harmonic Networks: Deep Translation and Rotation Equivariance". In : *CoRR* abs/1612.04642 (2016). arXiv : 1612.04642. URL : http://arxiv.org/abs/1612.04642.

[178] Liron YATZIV, Alberto BARTESAGHI et Guillermo SAPIRO. "O(N) implementation of the fast marching algorithm ". In : *Journal of Computational Physics* 212.2 (2006), p. 393 -399. ISSN : 0021-9991.

[179] J. Y. YEN. "An Algorithm for Finding Shortest Routes from All Source Nodes to a Given Destination in General Networks". In : *Quart. Applied Math* 27 (1970), p. 526-530.

[180] Hongkai ZHAO. "A fast sweeping method for Eikonal equations." In : *Math. Comput.* 74.250 (2005), p. 603-627.

[181] X. ZHUANG. "Decomposition of morphological structuring elements". In : *Journal of Mathematical Imaging and Vision* 4 (1994), p. 5-18.

[182] X. ZHUANG et R. M. HARALICK. "Morphological structuring element decomposition". In : *CVGIP* 35 (1986), p. 370-382.

Sixième partie

Annexe : sélection des articles

Fast and efficient FPGA implementation of connected operators

N. Ngan [a,*] , E. Dokladalova [b] , M. Akil [b] , F. Contou-Carrère [a]

[a] *Sagem, Massy-Palaiseau, France*

[b] *Université Paris-Est, Unité mixte CNRS UMR-8049, Computer Science department, ESIEE Paris, France*

Abstract

The Connected Component Tree (CCT)-based operators play a central role in the development of new algorithms related to image processing applications such as pattern recognition, video-surveillance or motion extraction. The CCT construction, being a time consuming task (about 80% of the application time), these applications remain far-off mobile embedded systems. This paper presents its efficient FPGA implementation suited for embedded systems. Three main contributions are discussed: an efficient data structure proposal adapted to representing the CCT in embedded systems, a memory organization suitable for FPGA implementation by using on-chip memory and a customizable hardware accelerator architecture for CCT-based applications.

Key words: Image processing, connected component tree, connected filters, graph, hardware implementation, FPGA, embedded architecture.

1 Introduction

Nowadays, speaking about the design of modern architectures for computer vision systems automatically means addressing the performance and the reuse problem. Algorithms in computer vision systems are asked to furnish a high performance and be capable of supporting the entire processing chain: from low-level to high-level processing in various applications. This is a never-ending problem in hardware design. If the performance is achieved by an optimization effort which means high system specialization, it will (by definition) limit its flexibility.

* Corresponding authors: Nicolas Ngan, nicolas.ngan@sagem.com

Obviously, the source of the problem is computational disparity: an application needs a variety of mathematical and algorithmic domains. It involves respecting different computing models. To overcome this problem, many researchers investigate the domain of reconfigurable/adaptable computing in order to find the alternative to the performance/flexibility trade-off [15]. Nevertheless, they remain optimized, in the majority of cases, for low level image processing [29, 14]. Several generalized coarse-grained systems can be adapted to high level processing [16, 24].

In our research, we study if the unified formalism which uses the tools from computational geometry (connectivity, data adjacency and proximity, graph transformation, etc.) allows the algorithm disparity problem to be overcome. We focus on the applications using **graph theory** as they could allow to bridge the gap between low-level [38, 43] and high-level [8, 39] processing implementations. In addition, many useful application domains are organized around graphs: image processing, data and knowledge bases, CAD optimization, digital problems, simulations.

In this paper, we focus on image processing operators using the Connected Components Tree (CCT). CCT plays a central role in the development of new algorithms related to image processing problems [35]. From the practical point of view, the advantage of these methods is that once the CCT is constructed, the processing is performed on the tree by graph transformation(s), and only one data structure is used from low-level to high-level processing (Fig.1). In addition, the graph transformations are applicable to any dimension (1D, 2D, 3D, ...).

Note that these algorithms have been successively used for filtering [20, 38], motion extraction [38] and watershed segmentation [31]. Besides, numerous practical applications exploit CCT data representation: pattern recognition of astronomical images [17, 4], microscopic image analysis [10], video-surveillance applications [34], image registration [23], data visualisation [6].

Fig. 1. Three main phases of connected component tree-based application

On the other hand, time efficient **CCT construction** is a challenging problem. It consumes about 80% of the application execution time[1], which is penalizing for a lot of the mentioned practical applications. This is because the graph structures can be very large and, for each operation, the applications need to access a large, data-dependent portion of the graph. Several

[1] It is a mean estimation obtained by profiling of two CCT construction methods published by P. Salembier [38] and L. Najman and M. Couprie [32]

algorithms have been proposed in order to solve the problem. In the majority of cases, they remain sequential and algorithm improvement relies on fast data structures (fifo-like) [38] or on optimization of computational complexity [22, 6, 8, 5, 42].

The problem of efficient connected component labelling have attracted the attention of hardware designers for many past decades [1, 30, 18, 19, 13]. However, to our knowledge, no efficient hardware implementation has been proposed so far (up to recently in [33]) for connected component tree algorithms due to its complexity. If any parallelization effort has been made on shared-memory computers by [25, 43, 21], there are few studies targeting hardware acceleration of computing CCT. Among them, we should mention Associative nets [27] allowing direct CCT computation or multi-FPGA Step-Based Architecture [11] where the authors present a distributed computing system based on a general graph. Nevertheless, these solutions are not ready yet for mobile embedded applications.

This paper presents an efficient FPGA implementation of CCT based computing suited for embedded systems. The three main contributions of the proposed concept have been validated on an FPGA platform:

- Adaptation of the CCT **data structure** allowing efficient **data mapping** in memory
- Proposal of an **on-chip memory organization** suited for CCT processing
- Proposal of **a hardware accelerator architecture for an embedded system** for CCT based applications

The paper is organized as follows: Section 2 *Connected Component tree algorithms* presents the algorithmical issues of the CCT-based applications. Here, we also discuss the data structures used to represent CCT and the adaptation of CCT allowing efficient data mapping in the FPGA memory. We focus on *Computing tasks dependency* analysis in Section 3 followed by a proposal of an efficient *Hardware implementation* in Section 4. Finally, the *Experimental results* are summarized in Section 5.

2 Connected Component Tree algorithms

CCT operators work on flat zones (connected components) of images, rather than on individual pixels. We call a connected operator an operator merging only flat zones. Hence, it cannot introduce any new contour. It simplifies and preserves the contour information [36, 9]. In the context of segmentation techniques, such as region growing or watershed, they rely on iterative merging strategies [28]. The CCT construction requires a global image analysis in order

to respect the relationship of the adjacent flat zones.

2.1 Basic mathematical notions

Let us consider a 2D greyscale image $f : D \to \mathbb{Z}$, $D = \mathbb{Z}^2$ and suppose that D is equipped with the neighbourhood mapping $N : D \to \mathscr{P}(D)$. $\mathscr{P}$ denotes the power set (the set of subsets). Then, $\forall x, y \in D$, we say that x and y are connected if $x \in N(y)$. Hereafter, let us assume that N is the 4-neighbourhood N_4, i.e. the set of the north, south, east and west neighbours. The associated connectivity is therefore the 4-connectivity.

For some $A \subset D$, we say that A is a connected set if $\forall a, b \in A$ there is a sequence of points $(a = x_1, ..., x_n = b) \subset A$, such that $x_{i+1} \in N_4(x_i)$, $1 < i < n$. For some A, $CC(A) \subset \mathscr{P}(A)$ shall denote the set of connected subsets of A.

For some $k \in \mathbb{Z}$, the set $C^k = \{x \mid f(x) \geq k\}$ is the set of points above the threshold k. The set $\{c_i^k\} = CC(C^k)$ denotes the set of connected components above k. For some $k, l \in \mathbb{Z}$, and $k > l$, it holds that $C^k \subseteq C^l$. The set of all connected components is called $C = \cup_k C^k$.

Let us consider a graph $G = G(V, E)$ where V, such as $V \to C$, is the set of nodes and $E \subset V \times V$ the set of edges. The nodes are tied by the relation child-parent. For some $v_i, v_j \in V$, there is an edge $e_{ij} \in E$ if $c_i \subset c_j$ and there is no c_k that $c_i \subset c_k \subset c_j$. We say that v_j is the parent of v_i and v_i the child of v_j. Notice that the graph is oriented, i.e. $e_{ij} \neq e_{ji}$.

Note that such a graph is an acyclic, oriented graph called **connected component tree**. There are special nodes called *leaves* and *roots*. The leaves do not have any children and the root does not have any parents.

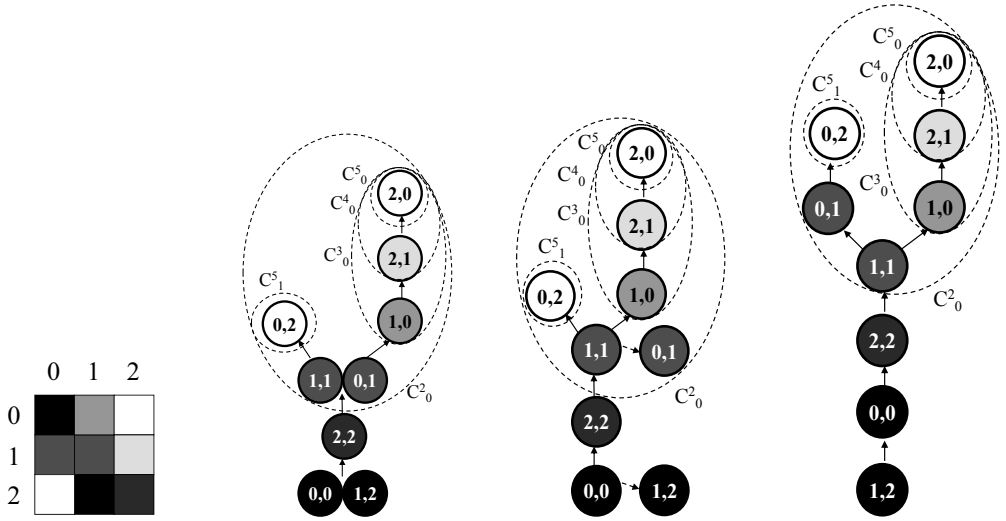
Due to the inclusion relation on the connected components, the graph nodes are organized in a tree structure suitable for filtering or segmentation [40].

2.2 CCT representations

To represent the CCT structure in memory, different data organization leading to different memory requirements, have been proposed : i) **"Native" CCT** (Figure 2(b)) [38] or ii) its "compressed" version **Canonical CCT** (Figure 2(c)) where one node point represents a component and each pixel belonging to a component points toward its representative node [32]; or **Point Tree** (Figure 2(d)), the burst version of the CCT where each node of a point-tree

represents a pixel in the picture [5]. 1-D tree computation is a special case [26], where intermediate results have to be gradually merged as described in [43].

According to the definition of c_i^k, some set of connected components are shown in Figure 2 for different tree structures. In this figure, connected components from level two to five are shown (encircled) as an example picture with six grey levels.



(a) Input image (b) Native CCT (c) Canonical CCT (d) Point Tree

Fig. 2. Illustration of different data structures

The reader can refer to Table 1 for evaluation of memory requirements. Based on the algorithm definitions, the memory requirements are roughly estimated as the sum of memory allocated for input/output and the overall working memory needed for intermediate results. Those estimations have been discussed in [21, 4]. Table 1 also contains the estimation of algorithm complexity. For the sake of clarity, we consider the classical $\mathscr{O}$ notation, where the worst-case scenario is given for a considered algorithm. Data type column means the data type for an input image pixel coding allowed by the algorithm.

Table 1
Comparison of algorithms: n is the total number of pixels in the image, G is number of grey levels of the image, α is a very slow-growing diagonal inverse of Ackermanns function [8].

Tree type	Complexity	Data type	Memory requirements
Native CCT [38]	$\mathscr{O}(nG)$	int	$4n$
Canonical CCT [32]	$\mathscr{O}(n\alpha(n))$	int, float	$6n$
Point tree [5]	$\mathscr{O}(n\log(n))$	int, float	$4n$
1-D [26]	$\mathscr{O}(n)$	int, float	n

2.3 Data structure adaptation: Parent Point-Tree

The data structures proposed in the above mentioned theoretical papers has not been designed for an efficient FPGA implementation. Neither is the number of nodes nor the size of each node, known beforehand and one has to manage complex memory allocations. This difficulty is evident in the case of native CCT structure or canonical CCT since the number and the size of the nodes depend on the image content.

From this point of view, the Point Tree (PT) presented in [5] seems to be an appropriate structure due to a fixed number of nodes. Originally, PT is an oriented graph where the parent node points toward the child node (Figure 2(d)), note that this is a common orientation of the discussed structures (Figure 2). In such a case, the issue is that the number of child nodes for each parent node can only be known at the end of the tree construction. Thus, the number of link memory allocations for each node is not predictable. As shown in Figure 3, we propose to reverse the PT orientation and we call it **Parent Point Tree (PPT)**. Thus, each node has at most, a unique parent node. Consequently, we only need to store the parent address for each child node to create tree connections and the number of nodes is equal to the number of pixels in the image.

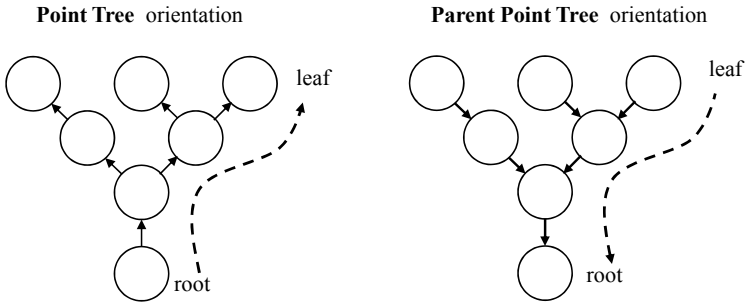


Fig. 3. Parent Point Tree orientation

The next section presents the PPT construction algorithm followed by its FPGA implementation.

2.3.1 Parent Point Tree construction algorithm

The proposed Parent Point Tree construction algorithm (Alg. 1) is a combination of Tarjan's Union-Find [40] as used in [5] with an acceleration technique inspired by the Najman-Couprie algorithm [8].

We define a *branch* as a subset representing a group of nodes that are connected to each other in the complete tree. A *branch* can then be composed of only one node or several nodes. A node which represents the extremity of a

branch (i.e. a node without any child nodes in the tree) is called a *leaf*. As the tree is gradually built with leaves, we call *local root*, the temporary root node for each branch that is not yet connected to the main tree.

We recall that in a *Point Tree* or a *Parent Point Tree*, a node is associated to only one pixel in the picture. Then, let us consider p a pixel of the image f, and q an adjacent pixel of p according to the considered neighbourhood $N_4(p)$ such as $q \in N_4(p)$.

In this algorithm, the tree is gradually built with leaves to the root. Consequently, it starts by building several *branches* that are not necessarily connected to each other. They become a partial tree which then contains several *local roots*.

The tree construction is divided into two parts: the first part involves the sorting of all the pixels in the image in increasing or decreasing order, depending on the type of tree. In order to build a *Max-Tree*, pixels are sorted in decreasing order and the leaves then represent the highest pixel values. The second part is building the tree by processing those pixels sequentially in this order. The algorithm presented in Alg. 1 corresponds to that second part, assuming a correct sorting of pixels p in a *queue* memory. The most costly part of the algorithm is to find the local root in order to link the components. At each neighbour node, it needs to explore the tree until it finds the root.

As described in Alg. 1, each pixel q from the neighbourhood N_4 of a pixel p is processed. The first step is to check its processed status. If it has already been processed from the pixel queue, it means that this pixel already belongs to a branch from partial tree (*tree_links_parent*). The aim is to find the local root of the branch where the neighbour pixel q belongs, to link the current pixel p to it (function *Link*). The following procedure is to jump from node to node in the branch (function *Jump*), starting from pixel q, and check successively if the node is the local root of the branch. As a consequence, this loop is very time consuming and depends on the branch length.

Thus, we propose to accelerate this process by gradually linking the current local root during the construction for each node in a fake and temporary tree so that the next access would point directly to it. We then create a compression of the tree. However, this technique is costly in terms of memory because we need to allocate a "mirror copy" of the original *tree_links_parent* tree. This copy called *tree_links_root* is stored in a memory only dedicated to the tree construction and it can be modified at will for accelerating the construction.

This shortcut technique works mainly in the *tree_links_root* memory by first checking the last registered local root node associated to the pixel q (function *Check*). If this registered node is not a real root node, then we have to jump from node to node starting from this last registered node in the compressed

tree to find the root. During that searching process, encountered nodes are stored in a temporary buffer (function *Store*) to be all updated by linking them to the last root node found (function *Update*). This update then accelerates the searching process for future node access.

When any pixel is linked to its local root, we mark it as *processed*.

The following operations are necessary to build the PPT:
Check(x,T) returns the local root of the node associated with pixel x in the tree T,
Jump(x,T) returns the parent node of the node associated with pixel x in the tree T,
Store(x) stores the current parent node associated with pixel x to be updated in buffer,
Link(x,y,T) links the current node associated with pixel x to the local root node for pixel y in the tree T,
Update(x,T) updates the local root node x for all the parent nodes in buffer for the tree T.

Algorithm 1 Parent Point Tree construction

Require: *queue*: pixels ordered in decreasing order
Ensure: Root-Stored PPT

```
while (queue not empty) do
  for all (q ∈ N4(p)) do
    if (q == processed) then
      root ← Check(q, tree_links_root)
      while (root not found) do
        parent_node ← Jump(q, tree_links_root)
        q ← parent_node
        Store(parent_node)
        root ← Check(q, tree_links_root)
      end while
      Link(p, root, tree_links_parent)
      Update(root, tree_links_root)
    end if
  end for
  p <= processed
end while
```

Note that the construction algorithm is sequential and its execution is data-dependent. The more complex in terms of components the image is, the longer the algorithm execution will be. A noisy picture for example, produced directly from an image sensor is represented by a more complex Point Tree in terms of number of leaves because of numerous un-representative high pixel values caused by the noise. Then, one possible solution is to pre-process the picture (low-pass filters such as denoising, smoothing filters) to remove those unuseful

small high values.

2.4 CCT filtering

In the Introduction, we have presented the general scheme of the CCT-based application (Fig. 1). Once a tree construction is completed, the filtering step is performed. In practice, the filtering step analysis each node and it evaluates some criteria [38]. It makes a decision whether the node is preserved or removed.

In this paper, an attribute denotes supplementary information associated with each node allowing to measure a given criterion. We can quote for instance some classical criteria: area, height [12], opening by reconstruction or λ-max operators [41] and other examples can be found in [36].

When the complete CCT is available, with associated node attributes, the filtering can begin. This step is also called CCT pruning [37], the branches of the tree structure are removed or preserved according to the *decision rule*. In this paper, we use the *direct decision rule* as defined in [38].

Let us consider the maxima of the image represented by the leaves of the CCT and some parameter λ. Starting from the leaves, we scan all the sequence of their ancestors going down to the root. The examined node is only preserved when the associated attribute value is higher than λ. Otherwise, the node is removed. When one node is removed, its content is merged with its nearest preserved ancestor.

3 Computing task dependency

As illustrated in Figure 4, we can see that there are four steps to complete an application. It starts with pixel sorting. The tree construction can only begin when all image pixels are sorted. In general, attribute computation can be done parallel (AC1) to the tree construction. Hence, the pixel values accessed for tree construction can be immediately reused for attribute computation. It allows to minimize memory accesses and to parallelize the computing tasks. If it cannot be done in parallel, it has to be executed after the complete tree construction (AC2). Only after that, the tree filtering can be done.

We might want to start the tree filtering process parallel to the tree construction by using attributes (i.e. height) that can be computed along with the construction. It is actually feasible when a pixel node has been identified as a leaf of the tree. The partial branch is read and could be processed step by step

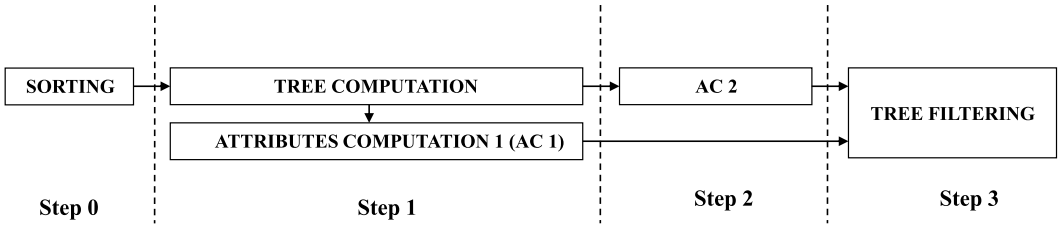


Fig. 4. Execution steps of a complete tree-based application

during its construction. However, it requires switching between branches during the tree construction as they are built gradually starting with the leaves. It obviously results in higher memory requirements to stand by the filtering process because branches are not always complete. As a consequence, it is more efficient to start the filtering process after the complete tree construction and the attributes computation as depicted in Figure 4.

4 Architecture proposal for embedded system

The global architecture (see Figure 5) of the system reflects the four presented execution steps. Hence, it consists of four main computing blocks: pixel sorting block, tree construction block, attribute computing block and filtering block.

For readability reasons, we indicate the address datapath by the symbol @ in Figure 5.

The main controller supervises an application execution and it manages the data flow between the computing blocks and the memory system. Finally, the computing blocks are interconnected by the switch fabric allowing the memory address and data paths to be redirected dynamically.

As shown in Figure 5, the global control is provided by the main controller which is composed of three command units : a block selector (which can activate or deactivate a computing block reporting its status -busy or not), a multi-switch controller (which can dynamically modify the Switch Fabric containing several mutiplexers) and a sequencer (which orders computing block activations and datapath modifications).

Thus, each computing block receives commands from the main controller. It contains an intelligent core: a controller that sequences local computing block operations.

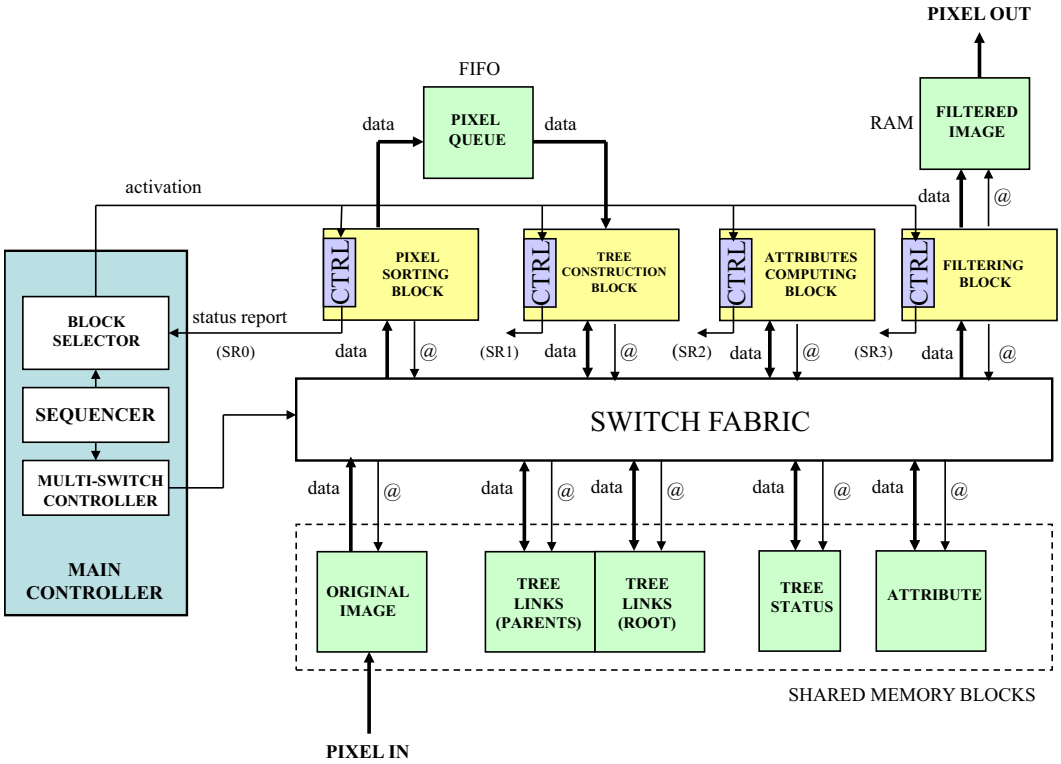


Fig. 5. Global architecture

4.1 Switch Fabric

In order to allow the memory address and data paths to be redirected dynamically, all the blocks share the memory and are plugged into the backbone of this architecture. The switch fabric is composed of selectors like multiplexer and demultiplexer basic components which are controlled by a multi-switch controller inside the main controller. It is therefore configurable before implementation and programmable after synthesis. The Switch Fabric is configured to the correct data path dynamically. Figure 6 shows one configuration example at runtime. The dashed lines represent data paths that are disabled at a specific time in the application.

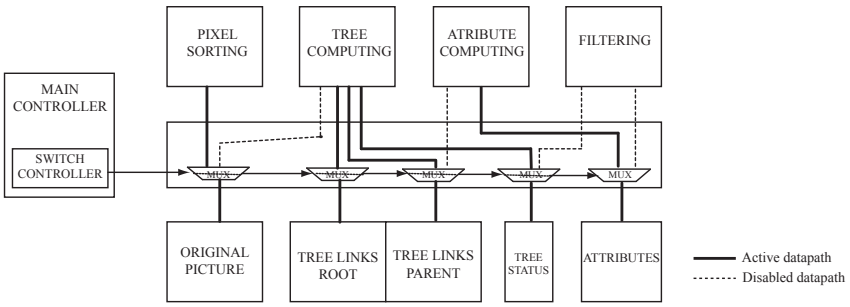


Fig. 6. Switch Fabric

4.2 Memory system

A CCT application works with six main types of data: original and filtered image, pixel queue, CCT, attributes and construction buffers needed during the processing (the tree status for instance). The designed memory system is based on the principle of shared memory constituted by several small independent memory blocks that can be accessed in parallel due to the Switch Fabric. By combinating those memory blocks to the Switch Fabric, we are able to reuse the memory blocks dynamically. However, we have added a dedicated memory for the pixel queue and the output image to buffer pixels more efficiently. We recall that the tree construction time is not deterministic and depends on the image content (Section 2.3.1).

At the beginning of the processing, the filtered image memory section is the exact copy of the original. The filtering is based on the built component tree. Depending on the given application (a segmentation, for instance with pixel extractions or deletions), we only modify the filtered image memory in the desired pixel position. A pixel queue memory (or pixel address queue) is a simple FIFO structure storing the output of the pixel sorting block. In fact, this memory has to store the different pixel positions in increasing or decreasing order.

The input image, CCT, attribute and construction buffers are placed in the shared memory. The original picture memory has to be shared with the other blocks because it contains the greyscale pixel value information contrary to the filtered image memory which is reserved to the filtering block. CCT is constructed in the link memory storing only the child/parent node relationships and the original picture is necessary for navigating between different components in the tree. Then, we have a part of memory dedicated to storing the different attributes (the height, the volume, etc.) as the output of the attribute computing block. During all the processing, some information might need to be buffered like the pixel positions for pruning the tree (the filtering process) by modifying the value or additional buffered addresses to eventually accelerate the processing. Finally, we need to temporarily store the status of each pixel node during the tree construction. We can call it "Tree status memory" and it contains bit flags which are detailed in Section 4.4.

4.2.1 Memory size

Let us consider an image with a width of M and a height of N. The pixel size is fixed to k bits. $ADDR\_SIZE$ is defined as equal to $ceil(log2(M) + log2(N))$ ($ceil()$ function rounds to the next largest integer). The memory sizing is presented in Table 2.

Table 2
Memory size (qSIF = 160x120 test image format); H=15 (height attribute); k=8

Storage element	Memory space (bits)	qSIF Application (bits)
Original image	$(2^{ADDR\_SIZE}) \times k$	262144
Filtered image	$(2^{ADDR\_SIZE}) \times k$	262144
Pixel queue	$ADDR\_SIZE \times M \times N$	288000
Tree links 1^{st} part (parents)	$2^{ADDR\_SIZE} \times ADDR\_SIZE$	491520
Tree links 2^{nd} part (roots)	$2^{ADDR\_SIZE} \times ADDR\_SIZE$	491520
Tree status	$(2^{ADDR\_SIZE}) \times 3$	98304
Attributes	$(2^{ADDR\_SIZE}) \times H$	491520

4.3 Pixel sorting block

The pixel sorting block operates on the original picture and fills a pixel queue for the tree construction block. This queue can be considered as a FIFO. The tree construction block computes the different links of the tree and stores them in a memory (tree link memory) which is shared with the other blocks (attribute computing block and filtering block in particular).

Let us take a simple example like a 3x3 picture with five levels of grey. For readability reasons in the following figures, each pixel is designated by a letter instead of its coordinates as shown in Figure 7(a).The corresponding parent point-tree of this example is presented in Figure 7(b).

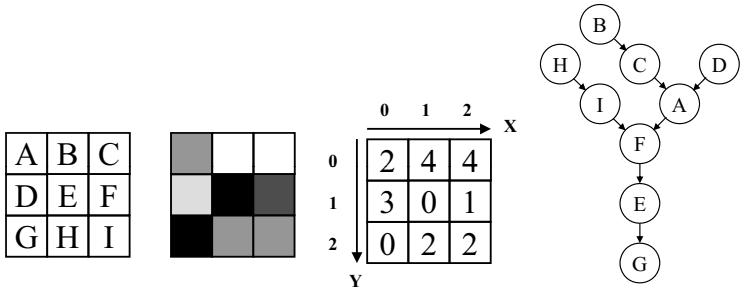
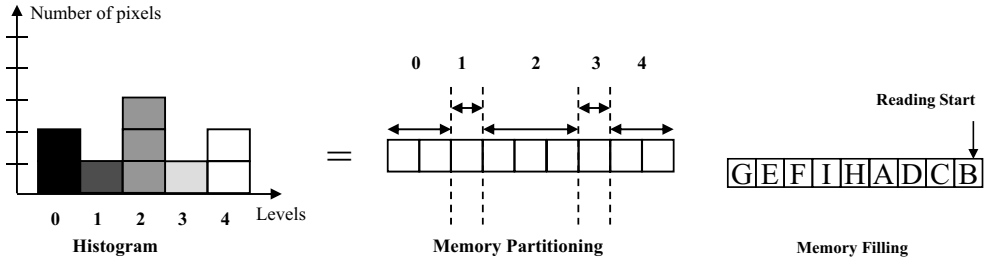


Fig. 7. 3x3 picture example

As mentioned in the previous section, pixel sorting is not the "time consumer" procedure compared to the creation of the connections between nodes (pixels in the parent point-tree). However, it can create high memory requirements. To avoid high consumption of memory blocks, we choose to use a counting sort method [7]. This technique is divided into two passes: the queue partitioning pass and the queue filling pass. In the first pass, we want to partition the queue memory by level sector and each size of the sector can easily be determined

by building a histogram (Figure 8(a)).



(a) First pass (b) Second pass

Fig. 8. Counting sort method

In our example, we can see that the number of pixels in each value is equal to the number of slots reserved in the queue memory. This pass can be done "on the fly" during the picture storage in the image memory. In the following pass, the picture is read from the image memory and the pixels are distributed in the queue memory according to their value.

In each level, the pixels are ordered according to the direction of reading. This order has an impact on the final parent point tree connections between the pixel nodes but it does not change the structure of the corresponding component tree when the pixels are grouped by component (See set c_i^k in Figure 2). In our example (Figure 8(b)), the reading of the queue starts with pixel B when we want to build a Max-Tree (decreasing pixel value order) or with pixel G for a Min-Tree (increasing pixel value order).

4.4 Tree construction block

The tree construction block implements the algorithm 1 presented in Section 2. As illustrated in Figure 9, this computing block contains five main processing units that can be associated to specific functions in the algorithm. The functions *Check(x,T)* and *Jump(x,T)* can be associated to the *Root Finder* unit returning the parent or root nodes. This unit works with the *Bit checking* unit by testing the status bits, corresponding to the if and *while* conditions in the algorithm. The function *Link(x,y,T)* is associated to the *Linker* unit by writing the origin pixel considered as the new local root in the *tree link parent* memory. The *Store(x)* and *Update(x,T)* functions corresponds to the *Root Updater* unit storing nodes in a temporary buffer to update them in the *tree links root* memory. Finally, the *Neighbour Addresses Generator* unit computes the adjacent pixels $q \in N_4(p)$ memory addresses.

The pixel sorting block provides the input data *queue* through the pixel queue containing all the pixel addresses ordered with respect to the decreasing pixel value. As shown in Figure 9, the neighbour address is verified with the Bit

Checking and the Root Finder units to get the status of the pixel node and to eventually get any parent address. If the bit status for the current pixel neighbour is *processed*, the Bit Checking unit notifies the Root Finder unit to loop. The parent address is then loaded to the current neighbour address register in order to get its parent pixel. This parent adress is also stored in the FIFO Buffer of the Root Updater unit (= *Store(x)* function). The checking loop ends when the bit status for the parent presence is *un-processed*. It means that we reach the current root in our tree under construction. Consequently, the Linker unit can update the identified current root by the origin pixel address (= *Link(x,y,T)* function). The origin pixel node becomes the new pixel parent node and its status bits are updated. All buffered pixel addresses that have been met, are updated to be linked to the origin pixel node (= *Update(x,T)* function). Thus, that node becomes the latest local root and the tree construction block can request the following origin address from the pixel queue (*queue*).

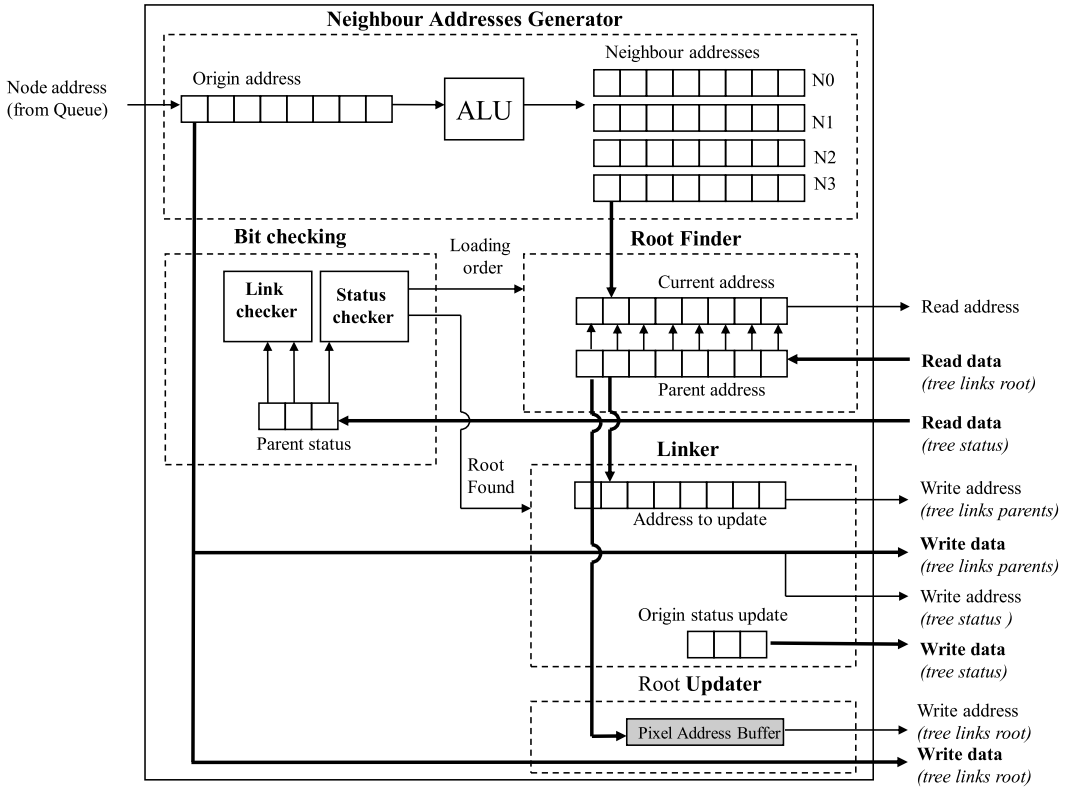


Fig. 9. Tree construction block

Tree construction example: Each pixel node has three bit flags: "Is Processed" (IP) bit to check if the pixel node has been processed, the Parent Checking (PC) bit, meaning that the pixel node has a parent node and the Child Checking (CC) bit, meaning that the pixel node has at least one child node. Let us begin with the pixel B whose coordinates are (1, 0) in Figure 7(a).

This pixel is located on the edge of the picture and its neighbours $N_4(B)$ are C, E and A. According to the pixel coordinates, nonexistent pixel neighbours (represented by XXXX in Figure 10) are just skipped from analysis in the Root Finder knowing the image dimension.

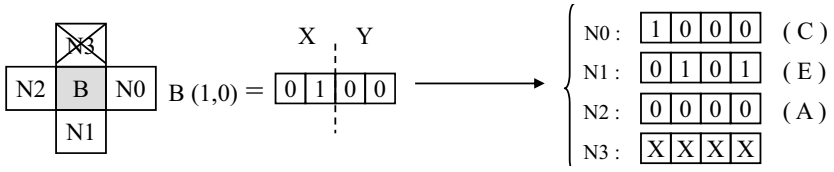


Fig. 10. Pixel B neighbours

At the beginning, none of the pixels are processed and all the "*processed*" - IP bit flags in the tree status table are down. Consequently, all pixel B neighbours are not processed yet. Pixel B becomes a leaf of the tree and is temporarily linked to itself meaning that it is a current root of the tree in construction. Then, the IP bit flag for pixel B is up (Figure 12(a)) and the tree construction block can request the following origin pixel in the queue which is pixel C in our example.

When pixel F is reached (Figure 11), some pixels have already been processed (B, C, D, A, H, and I - See pixel queue in Figure 8(b)) and they might be linked to this pixel. Thus, we are only interested in processing the neighbour N1 (I) and N3 (C) (Figure 11)

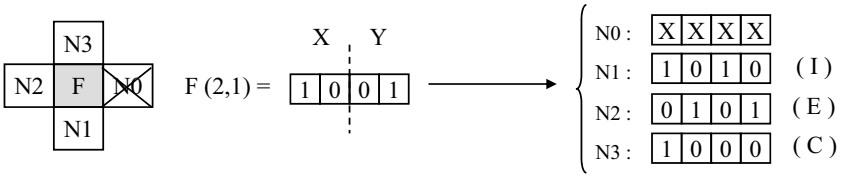
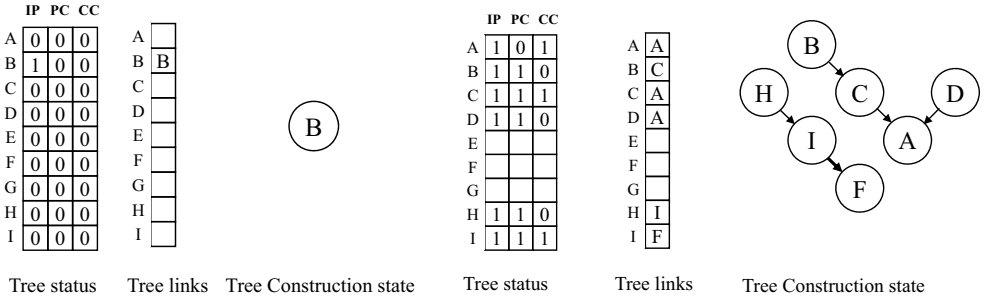


Fig. 11. Pixel F neighbours

Pixel I is already a current root and it is directly linked to pixel F by updating the tree status table (the CC bit to 1) and the tree link table (F address for pixel I).

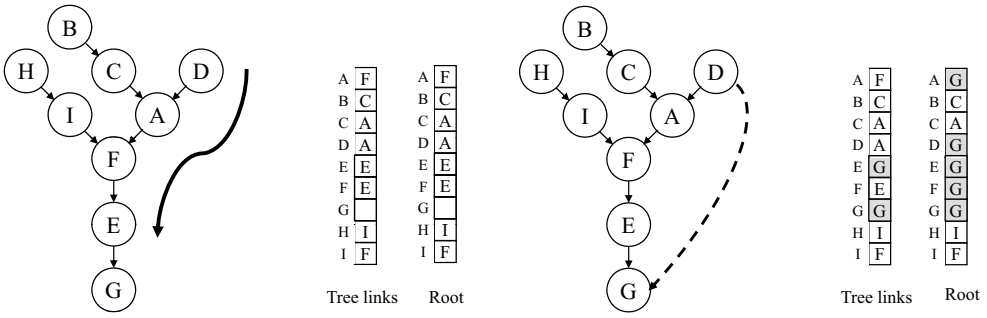
Pixel C is not a current root because its PC bit flag is up. The Root Finder unit in the tree construction block replaces the current address (pixel C) by its direct parent address (pixel A). Pixel A is a current root because its PC bit flag is down (Figure 12(b)). After linking it to pixel F, its tree status bits will be updated from [1 0 1] to [1 1 1].

Shortcut technique example: Let us illustrate the shortcut technique by a simple example. From the Figure 13(a), let us assume that we want to link the G node in the tree beginning with its neighbour D node. The current root E is therefore linked to itself for the time being.



(a) Pixel B (b) Pixel F

Fig. 12. Tree tables



(a) Before linking (b) After linking

Fig. 13. Shortcut technique example

The tree is explored starting with the D node and the nodes A, F and E are met before linking. When the "finding root process" is triggered, all the nodes that have been met during the exploration, are linked to the pixel point to be attached. In other words, the origin pixel will be the current root of the tree branch. If we consider that this root becomes the parent of all the pixel nodes that have been met, the following accesses would be faster by making large node jumps. In our example, we can store the nodes A, F and E in a temporary memory and update their "virtual" parent node G in the root memory as shown in Figure 13(b). Note that the exploring accesses are consequently made on the root memory.

4.5 Attribute computing block

As mentioned in section 2.4, attributes are additional node characteristics that can be used for filtering an image depending on user criteria (conditions on height, area, depth or volume of a component for example). Some of them can be computed during the tree construction and others have to be computed when the tree is complete. The depth of a node can be computed, for instance, on-the-fly with a counter and additional logics (min-max) during the tree construction.

Depending on the complexity of required attributes, the computing block might be time consuming. The study of attributes is beyond the scope of this paper and one can find some attribute computations in [25, 32].

Note that an attribute computing block is not mandatory to complete an application. In order to save memory and time, the simplest implementation can be done without attributes but the criteria have to be chosen so that they are merely based on components and pixel values. Some basic criteria could amount to conditions on the contrast between two components by subtracting consecutive component levels. Nevertheless, computing attributes allows to apply more complex criteria on the tree for better results in the filtering process.

4.6 Filtering block

The component tree can be used as a new working base for filtering the associated picture. Thus, pixels or components of the picture can be manipulated by changing the value or the node positions in the tree. The component tree is particularly interesting compared to the classic 2-D matrix picture representation because it gives the hierarchical layer disposition, in addition to the spatial pixel position. Traditionally, a greyscale picture can be viewed as a set of hills (highest values) and valleys (lowest values) with a layer organisation. Flattening hills means removing the highest layers and leaving the last implied underlayer apparent.

In a tree-based application, a picture is usually simplified by pruning the corresponding component tree. Back in Figure 2, pruning the tree at C_0^3 is removing the upper nodes (2,0) and (2,1) to leave the underlayer value apparent (level 3). Thus, pruning the component tree can be understood as flattening hills. Note that a tree pruning application can be found in [4] and [5] for an astronomical context where the aim is to clean the sky from stars in greyscale pictures.

In our context, we propose an original tree-based filtering application based on the component tree pruning. Instead of merging components (i.e. flattening hills) with a traditional tree pruning process, we propose to give an arbitrary value for all the identified nodes to be removed from the tree. Thus, it results in an original local thresholding by using the component tree. It can be efficient for simplifying pictures from an infrared camera because the most interesting parts are the highest local values in the image. Those local parts can have different levels of grey and using an arbitrary global level threshold would be obviously inefficient.

Consequently, a specific tree pruning block has been designed for this purpose.

Flattening the hills of a greyscale picture can be done technically by replacing the removed pixel value nodes with the pixel value of their parent nodes. In our case, we replace any removed pixel value nodes with an arbitrary value.

A tree pruning process consists in exploring the tree from each leaf to the root in order to cut down branches when proposed criteria are not respected as explained in section 2.4. Those criteria are based on previously computed attributes (height, volume, depth, etc.) and by combining them, we can obtain original results. One can find some examples in [25].

The leaves of the tree are the starting points for the filtering process. They can be identified with the *Child Checking* (CC) bit from the tree status table. We recall that a CC bit flag down means that the pixel node does not have a child node. Notice that the reading direction of the tree, from its tops to its root, is well suited considering the proposed tree link structure (a parent-tree where child nodes point at its parent node) and the targeted application (local thresholding). Since we know that we need the leaves, they can be stored during the tree construction in a buffer (i.e. a FIFO for instance) to feed the tree pruning block. In particular, we can note that, during the tree construction, a current processed origin pixel node, which is not linked to any other nodes, becomes a leaf node.

The tree pruning block architecture is presented in Figure 14 with two criteria in input. It contains three main processing units: a *Tree explorer* unit, an *Attribute collector* unit and an *Image updater* unit.

Notice that, in addition to the *tree links parent* table, the block has to refer to the original image to identify the component values.

As illustrated in Figure 14, each leaf node address is loaded in the *Tree explorer* unit. This unit works parallel to the *Attribute collector* unit which reads successively the associated attributes for each processed node. The *Tree explorer* unit explores each tree branch by several reading loops in the *tree links parent* table until one of the criteria is not valid according the comparison units.

When the criteria are respected, the pixel node is stored in a temporary "Zone FIFO" buffer located in the *Image Updater* unit, until the end of a tree branch exploration. When one of the criteria is not respected, the branch is modified by overwriting the pixel value of the nodes stored in "Zone FIFO" with a specific value. Note that the use of this FIFO is optional in our case because we impose a value. Thus, the overwriting can be done gradually. The FIFO is only necessary for a tree pruning block that requires to merge components.

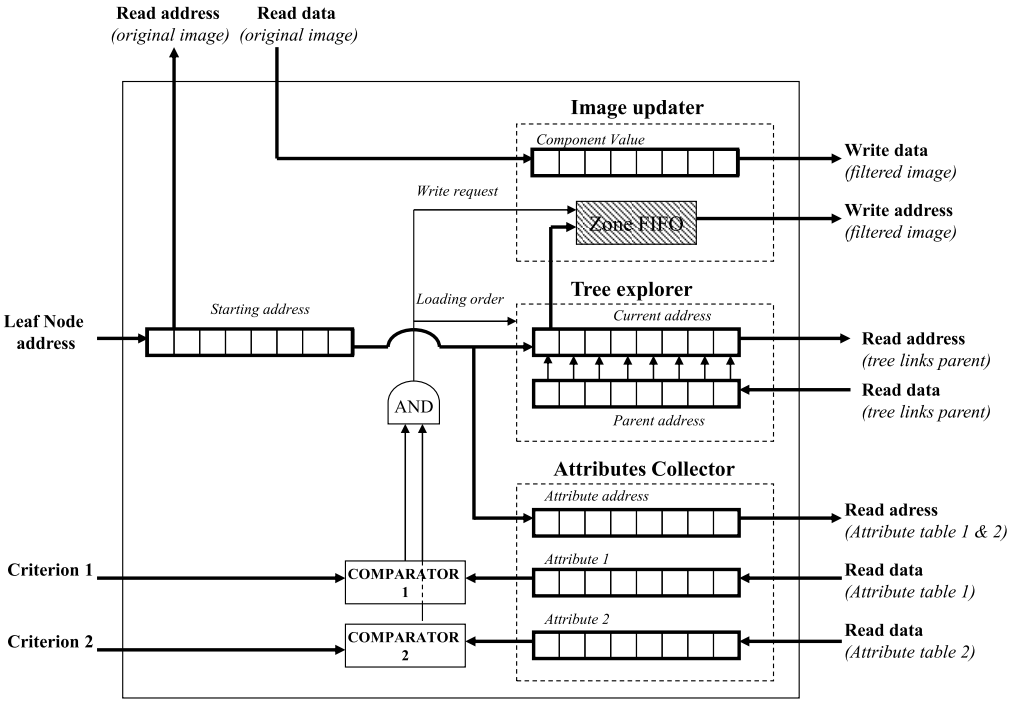


Fig. 14. Tree pruning block

5 Experimental results

The proposed generic system architecture has been implemented and validated on ALTERA Stratix II 2S60 Development board [2]. To the best of our knowledge, there is no other FPGA implementation of a complete application based on tree construction. For our hardware implementation, we propose a simple infrared (IR) hot spot image filtering application based on pruning the tree. To maximize the memory bandwidth, we choose to use the on-chip memory directly available on the FPGA chip. These RAM Memory blocks are very limited resources and in our case, we had a budget of 2,544,192 On-chip RAM bits for the 2S60 version. Moreover, Stratix II devices features a particular memory system called TriMatrix based on three sizes of embedded RAM blocks (M512, M4K and M-RAM) which can be configured in different addressing structures. Further information can be found in [3].

In particular, we used M4K (4 Kbits) and M-RAM (512 Kbits) memory blocks to fit our memory system and we used an input image resolution of 160x120 coded with 8 bits pixels. Thus, the different tables (Tree Links and Tree Roots) must have 15-bit-wide data words and the Tree Status table has 3-bit-wide data words (IP, PC and CC bits). Additional memory is needed for the pixel address queue, the two images (one for keeping the original picture and the other for storing the filtered picture) and several buffers (Root, Zone and Leaf

FIFOs).

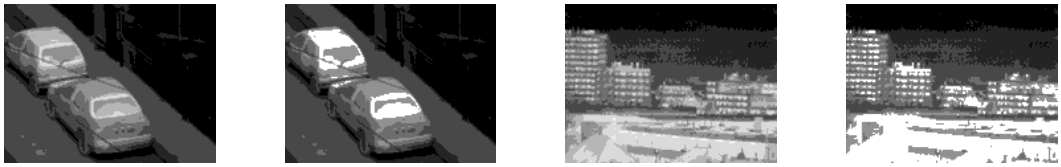
The FIFOs (Root, Leaf and Zone) size is defined according to the free memory space left. Smaller FIFOs decrease the overall performance. Note that it is possible to share the same FIFO for the Root FIFO and the Zone FIFO because they are not used at the same time. It is also possible to decrease the number of level counters by only taking the four most significant bits of the input pixel value.

One can notice that the Attribute Table is not really necessary for our first implementation. The pruning can simply be tested on the basis of pixel values and specified by an arbitrary threshold. Our focus is on the tree processing time and we want to validate our architecture conception.

Four types of pictures have been selected for the test (Fig. 15 and 16): a gradient picture with smooth grey level transitions (Fig. 15(a)) and three infrared (IR) image camera outputs.

(a) Gradient test (b) Gradient test result (c) Man IR test (d) Man IR test result

Fig. 15. Test images



(a) Cars IR test (b) Cars IR test result (c) Rooftop IR test (d) Rooftop IR test result

Fig. 16. Test images 2

Those IR pictures represent real scenes (man, cars, rooftop) with the different heat levels detected by the sensor. The man IR picture (Fig. 15(c)) represents the direct output from the image sensor and is a very noisy picture. The two others (Fig. 16(a) and 16(c)) represent pre-processed pictures. Those last two pictures, showing cars and rooftops, have been denoised by a median filter and the contrast has been enhanced by a histogram equalization. As a consequence, those conditions have a direct impact on the number of connected components to process and the tree structure complexity as shown in Table 3. In this table, the man IR picture is the most complex scene to process because of the number of connected components resulting from the high noise in the image sensor.

The results (Fig. 15(b), 15(d), 16(b) and 16(d)) represent the detection of the highest local components in the test pictures by combining attributes. In this case, we use an arbitrary max pixel value threshold λ applied to two attributes : contrast (component intensity difference) and height (number of pixel value component layers). It then creates a satisfying local thresholding for our prototype.

Table 3
Characteristics of test images

Image	Size	Number of connected components
Man	160×120	3068
Gradient	160×120	353
Cars	160×120	84
Rooftop	160×120	362

As shown in the pie (Figure 17), the architecture uses 78 % of memory space available and only 30 % represents the tree information (Original image and Tree Links). The remaining memory is dedicated to compute the tree, to accelerate the construction and to buffer intermediate data. Table 4 contains the results in terms of resource utilization obtained after the synthesis on the Stratix II FPGA. The logic utilization is very small as all the work is based on memory management, extremely important for the FPGA implementation. Note that our implementation has a maximum frequency of about 100 MHz for an Altera Stratix II FPGA EP2S60F484C4 target [2]. That maximum frequency is limited by the logic elements required to make the Switch Fabric. For the FPGA prototype, the system runs at 50 MHz as shown in Table 5 for measuring timing performance.

The 160×120 images are low resolution and come from a 320×240 IR image input downscaled by 2. That resolution is acceptable as it aims at low resolution display screens embedded in portable systems. The global architecture in Figure 5 shows that the memory, required for computing a picture, scales linearly according to its resolution. Thus, by keeping the same algorithm and the same architecture, there are only two options for a higher image resolution implementation. Firstly, to keep the memory performance, the simplest way is to upgrade the FPGA target for a higher on-chip memory capacity (such as Altera Stratix III or IV). Secondly, in order to keep the same FPGA target (for area constraint), the memory tables have to be stored in external memories. That second option greatly impact the timing performance because of the external memory access time and it also needs additional logics such as memory controllers. However, it remains relevant for applications without any critical real-time constraint such as photo applications which require image quality and higher resolution.

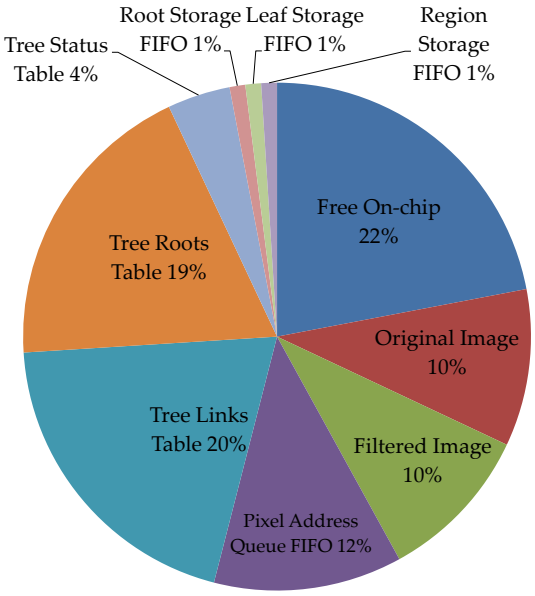


Fig. 17. On-chip memory block occupation in Stratix II 2s60

Table 4
Synthesis report STRATIX II 2S60

Resources	Quantity	Occupation 2S60
Combinational ALUTs	2680	6%
Logic Registers	3564	7%
Total RAM bits	1984470	78%

Table 5
Measured execution time (clock: 50 MHz)

Image	Size (pixels)	Tree construction (ms)	Filtering (ms)
Man	160×120	8.25	0.12
Gradient	160×120	7.41	0.22
Cars	160×120	7.75	0.55
Rooftop	160×120	7.93	6.55

Since the tree construction depends on the image content, the execution time is not known beforehand. This complexity is not only based on the number of grey levels and the connected components but also on their mutual relations (adjacency). The proposed algorithm is inspired from by already discussed quasi-linear algorithm [32], hence the execution times do not vary significantly even if the number of connected components is very different; see the Table 5 for the execution time for the tree construction. Note that in the test pictures (Fig. 15), even if the number of connected components for the Man test picture is nine times higher than the others, the tree construction

time overhead is just about 10 %. That performance results from the shortcut technique which allows to reduce the construction time dependency from the picture complexity. Note that even if the Gradient test picture has the lowest tree construction time, the shortcut technique is less efficient because the pixels are already spatially ordered and the Tree Root table is therefore less requested. Table 5 gives the execution time for the filtering. Note that the filtering time is longer for the Rooftop picture because more pixels are detected. Those timings depends on the tree complexity (number of leaves), the attributes used (contrast and height for this case) and the number of modified pixels (thresholding). Considering those variable parameters in practice, the complete application can run from 50 to 120 frames per second.

Finally, the proposed implementation allows to obtain an acceleration up to 3 on the Stratix II at 50 MHz for the tree construction compared to a standard desktop PC with Intel Pentium IV HT (3GHz) processor running under Linux. We obtain, for instance, respectively 16 ms and 21 ms for the Gradient and Man IR test images with Najman-Couprie algorithm on Pentium IV. Note that we use the C-implementation of the Salembier [38] and Najman-Couprie [32] algorithms for this comparison.

6 Conclusion

This paper presents an efficient FPGA implementation of CCT algorithms suited for embedded systems. The main contributions are: the adaptation of the CCT data structure allowing efficient data mapping in the memory, the proposal of an on-chip memory organization suited for CCT processing and a proposal of an overall embedded system for CCT-based application. After the introduction of state-of-the-art CCT algorithms and implementation, the design issues in terms of data structure and memory are discussed. The proposed architecture is then presented in detail. Assuming that this study is motivated by the search for a new approach to the flexibility of embedded systems, it demonstrates the FPGA implementation feasibility of relatively complex algorithms. In future, we will concentrate on the tree merging methods and implementation of the data parallelization by merging several 1D trees. Concerning the design challenges, the main objective is to explore and improve the interconnections between computing and memory resources.

References

[1] H.M. Alnuweiri and V.K. Prasanna. Parallel architectures and algorithms for image component labeling. *IEEE Transactions on Pattern Analysis*

and Machine Intelligence, 14(10):1014–1034, 1992.

[2] ALTERA. Stratix ii ep2s60 dsp development board data sheet. 2006.

[3] ALTERA. Trimatrix memory in stratix ii devices. 2006.

[4] C. Berger, T. Geraud, R. Levillain, N. Widynski, A. Baillard, and E. Bertin. Effective component tree computation with application to pattern recognition in astronomical imaging. In *ICIP07*, volume IV, pages 41–44, 2007.

[5] C. Berger and N. Widynski. Using connected operators to manipulate. Technical report, LRDE Seminar, July 2005.

[6] Yi-Jen Chiang, Tobias Lenz, Xiang Lu, and Günter Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Comput. Geom. Theory Appl.*, 30(2):165–195, 2005.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[8] M. Couprie, L. Najman, and G. Bertrand. Quasi-linear algorithms for the topological watershed. *J. Math. Imaging Vis.*, 22(2-3):231–249, 2005.

[9] José Crespo, Jean Serra, and Ronald W. Schafer. Theoretical aspects of morphological filters by reconstruction. *Signal Process.*, 47(2):201–225, 1995.

[10] O. Cuisenaire and E. Romero. Automatic segmentation and measurement of axones in microscopic images. In *SPIE Medical Imaging*, volume 3661 of *Lecture Notes in Computer Science*, pages 920–929. IEEE, 1999.

[11] Michael deLorimier, Nachiket Kapre, Nikil Mehta, Dominic Rizzo, Ian Eslick, Raphael Rubin, Tomas E. Uribe, Thomas F. Jr. Knight, and Andre DeHon. Graphstep: A system architecture for sparse-graph algorithms. In *FCCM '06: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 143–151, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Jonathan Fabrizio and Beatriz Marcotegui. Fast implementation of the ultimate opening. In *Proceedings of the 9th International Symposium on Mathematical Morphology and Its Application to Signal and Image Processing*, ISMM '09, pages 272–281, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] Holger Flatt, Steffen Blume, Sebastian Hesselbarth, Torsten Schunemann, and Peter Pirsch. A parallel hardware architecture for connected component labeling based on fast label merging. In *ASAP '08: Proceedings of the 2008 International Conference on Application-Specific Systems, Architectures and Processors*, pages 144–149, Washington, DC, USA, 2008. IEEE Computer Society.

[14] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, R. Reed Taylor, and R. Reed. Piperench: A reconfigurable architecture and compiler. *Computer*, 33:70–77, 2000.

[15] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *DATE01: Proceedings of the conference on Design, automation and test in Europe*, pages 642–649, Munich, Germany, 2001. IEEE

Press.

[16] Reiner W. Hartenstein, Michael Herz, Thomas Hoffmann, and Ulrich Nageldinger. Mapping applications onto reconfigurable kress arrays. In *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, pages 385–390, London, UK, 1999. Springer-Verlag.

[17] A.C. Jalba, M.H.F. Wilkinson, and J.B.T.M. Roerdink. Morphological hat-transform scale spaces and their use in pattern classification. *Pattern Recognition*, 37(5):901–915, May 2004.

[18] Shuenn-Der Jean, Chi-Min Liu, Chih-Chi Chang, and Zen Chen. A new algorithm and its vlsi architecture design for connected component labelling. In *International Symposium on Circuits and Systems (ISCAS)*, pages 565–568, 1994.

[19] Christopher T. Johnston and Donald G. Bailey. Fpga implementation of a single pass connected components algorithm. *Electronic Design, Test and Applications, IEEE International Workshop on*, pages 228–231, 2008.

[20] R. Jones. Component trees for image filtering and segmentation. In E. Coyle, editor, *Proceedings of the 1997 IEEE Workshop on Nonlinear Signal and Image Processing*, Mackinac Island, September 1997.

[21] P. Matas, Eva Dokladalova, Mohamed Akil, Thierry Grandpierre, L. Najman, M. Poupa, and V. Georgiev. Parallel algorithm for concurrent computation of connected component tree. In Jacques Blanc-Talon, Salah Bourennane, Wilfried Philips, Dan C. Popescu, and Paul Scheunders, editors, *ACIVS*, volume 5259 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2008.

[22] Julian Mattes and Jacques Demongeot. Efficient algorithms to implement the confinement tree. In *DGCI '00: Proceedings of the 9th International Conference on Discrete Geometry for Computer Imagery*, pages 392–405, London, UK, 2000. Springer-Verlag.

[23] Julian Mattes, Mathieu Richard, and Jacques Demongeot. Tree representation for image matching and object recognition. In *DCGI '99: Proceedings of the 8th International Conference on Discrete Geometry for Computer Imagery*, pages 298–312, London, UK, 1999. Springer-Verlag.

[24] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. In G. A. Constantinides P. Y. K. Cheung and J. T. de Sousa, editors, *Field-Programmable Logic and Applications*, pages 61–70. Springer, 2003.

[25] A. Meijster. *Efficient Sequential and Parallel Algorithms for Morphological Image Processing*. PhD thesis, University of Groningen, 2004.

[26] D. Menotti-Gomes, L. Najman, and A. de Albuquerque Araujo. 1d component tree in linear time and space and its application to gray-level image multithresholding. In Gerald Jean Francis Banon, Junior Barrera, Ulisses de Mendonça Braga-Neto, and Nina Sumiko Tomita Hirata, editors, *International Symposium on Mathematical Morphology*, volume 1,

pages 437–448. INPE, 2007.

[27] Alain Mérigot. Associative nets: A graph-based parallel computing model. *IEEE Trans. Comput.*, 46(5):558–571, 1997.

[28] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, september 1990.

[29] Takashi Miyamori and Kunle Olukotun. Remarc: Reconfigurable multimedia array coprocessor. In *IEICE Transactions on Information and Systems E82-D*, pages 389–397, 1998.

[30] Alina N. Moga and Moncef Gabbouj. Parallel image component labeling with watershed transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):441–450, 1997.

[31] L. Najman and M. Couprie. Watershed algorithms and contrast preservation. In *DGCI'03*, volume 2886, pages 62–71. Lecture Notes in Computer Sciences. Springer Verlag, 2003.

[32] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *IEEE Transactions on Image Processing*, 15(11):3531–3539, 2006.

[33] Nicolas Ngan, F. Contou-Carrère, B. Marcon, S. Guerin, Eva Dokládalova, and Mohamed Akil. Efficient Hardware Implementation of Connected Component Tree Algorithm. In *Workshop on Design and Architectures For Signal and Image Processing*, 2007.

[34] Patrick Piscaglia, Andrea Cavallaro, Michel Bonnet, and Damien Douxchamps. High level description of video surveillance sequences. In *EC-MAST '99: Proceedings of the 4th European Conference on Multimedia Applications, Services and Techniques*, pages 316–331, London, UK, 1999. Springer-Verlag.

[35] P. Salembier and L. Garrido. Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. *IEEE Transactions on Image Processing*, 9(4):561–576, April 2000.

[36] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *IEEE Transactions on Image Processing*, 4(8):1153–1160, August 1995.

[37] Philippe Salembier and Luis Garrido. Connected operators based on region-tree pruning. In Max Viergever, John Goutsias, Luc Vincent, and Dan S. Bloomberg, editors, *Mathematical Morphology and its Applications to Image and Signal Processing*, volume 18 of *Computational Imaging and Vision*, pages 169–178. Springer US, 2002.

[38] Philippe Salembier, A. Oliveras, and Luis Garrido. Anti-extensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4):555–570, April 1998.

[39] Pierre Soille. Constrained connectivity for hierarchical image decomposition and simplification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(7):1132–1145, 2008.

[40] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[41] Luc Vincent. Morphological area openings and closings for grey-scale

images. In *Proc. NATO Shape in Picture Workshop*, pages 197–208. Springer, 1992.

[42] M. Wilkinson and J. Roerdink. Fast morphological attribute operations using tarjan 's union-find algorithm. In *Mathematical Morphology and its Applications to Image and Signal Processing, Kluwer*, pages 311–320, 2000.

[43] Michael H. F. Wilkinson, Hui Gao, Wim H. Hesselink, Jan-Eppo Jonker, and Arnold Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(10):1800–1813, 2008.

Image Anal Stereol 2003;22:121-132
Original Research Paper

A PARALLEL ARCHITECTURE FOR CURVE-EVOLUTION PARTIAL DIFFERENTIAL EQUATIONS

EVA DEJNOŽKOVÁ AND PETR DOKLÁDAL

School of Mines of Paris, Center of Mathematical Morphology, 35, Rue Saint Honoré, 77 300 Fontainebleau, France
e-mail: {dejnozke,dokladal}@cmm.ensmp.fr
(Accepted May 20, 2003)

ABSTRACT

The computation of the distance function is a crucial and limiting element in many applications of image processing. This is particularly true for the PDE-based methods, where the distance is used to compute various geometric properties of the travelling curve. Massive Marching[a] is a parallel algorithm computing the distance function by propagating the solution from the sources and permitting simultaneous spreading of component labels in the influence zones. Its hardware implementation is conceivable as no sorted data structures are used. The feasibility is demonstrated here on a set of parallely-operating Processing Units arranged in a linear array. The text concludes by a study of the accuracy and the implementation cost.

Keywords: distance function, hardware for image processing, partial differential equations, parallel computing.

[a]Short preliminary study published in: Dejnožková and Dokládal (2003b) (algorithm) and Dejnožková and Dokládal (2003a) (architecture)

INTRODUCTION

Recently, the image processing methods based on Partial Differential Equations (PDE) have gotten an ever increasing attention. The examples of application can be found in numerous domains such as filtering (non-linear diffusion), active contours used for segmentation of either static images (Voronoï graph, watershed, shortest path, object detection) or sequences (object tracking) or more recent methods as shape from shading. The implementation of the PDE-based method requires the computation of non-linear functions. They are often solved by iterative and recursive algorithms characterized by a high computational cost. Therefore, only a limited number of real-time applications exists. The most common existing custom chips implement non-linear filtering, *i.e.* the non-linear diffusion (Perona and Malik, 1988; Gijbels *et al.*, 1994). One can find some experiments with super-computers (Sethian, 1996) or some examples of PDE-based segmentation using the level-sets implemented on graphic hardware (Rumpf and Strzodka, 2001).

The design of a custom chip becomes meaningful not only for the acceleration but also for the implementability on embedded systems (Suri *et al.*, 2002). Many authors put forth a considerable effort to reformulate existing sequential algorithms in a parallel form or to speed up the convergence (Weickert *et al.*, 1998). However the number of necessary iterations remains excessively high.

The motivation of our work is to define a general type of parallel architecture fitting the needs of the above-mentioned applications. There are several reasons why few architecture propositions have been made for interface (curve-based) algorithms. The curve traveling in a continuous space $\mathbb{R}^n$ is implicitly described in a discrete space $\mathbb{Z}^n$ by the distance to the curve, as proposed first in Osher and Sethian (1988). The distance function is the computation support of the level-set methods. Its zero-level set represents the traveling interface. Its accurate computation is very important because its geometrical properties (e.g. the curvature) are then used to describe its evolution in the time. The interface evolution imposes frequent and random memory accesses. Also, the numerical solution of a classical variational formulation leads to deformations of the implicit description of the curve (Kimmel, 1995) imposing more or less frequent reinitializations. In the work of Zhao *et al.* (1996) and Gomez and Faugeras (1992) can be found propositions of algorithm without re-initialization paid by the necessity to search the propagation speed on the zero-level set even for the points situated elsewhere. Because of the complexity and high computational cost, the classical re-initialization approach is still leading (Paragios, 2000). Repetitive re-initialization alternated with another type of processing increases the requirements on the implementing architecture.

Other applications, where the distance function is the result and not only a support, are the reconstruction

of 3D surfaces, minimal path search (Kimmel and Sethian, 2001) or continuous watershed (Meyer and Maragos, 1999).

In Gijbels *et al.* (1994) has been proposed a linear-array, SIMD architecture for PDE-based filtering by linear diffusion. An efficient hardware (parallel) implementation of a (weighted) distance function preserving the accuracy and the necessary sub-pixel precision for the continuous interface evolution is still needed. The goal of this paper is to show that the same architecture type can also be used for interface evolution algorithms.

This paper is organized as follows. After a brief review of existing distance computing algorithms, we analyse the principles of the Massive Marching. Section "Hardware implementation" presents its hardware implementation. Experiment results state the achieved implementation parameters such as the surface requirements, clock rate and necessary fixed-point precision for a given error.

REVIEW OF EXISTING ALGORITHMS

The sub-pixel accuracy is one of the essential features required from algorithms computing the distance function for the level-set methods. The initial condition, a closed curve $\mathscr{C}_0$ placed arbitrarily in $\mathbb{R}^n$, represents the zero-level set of the searched distance function $u(x,y)$. For $n=2$ we have:

$$\mathscr{C}_0 = \left\{(x,y) \in \mathbb{R}^2 \mid u(x,y) = 0\right\} . \qquad (1)$$

Recall that u is discrete, *i.e.* only $u(i,j),\ i,j \in \mathbb{Z}^2$ is known. Other necessary characteristics required from the distance computing algorithms are the preservation of the accuracy, computation on a narrow band for the active contours and simultaneous propagation of components labels.

Current algorithms used to find u given $\mathscr{C}_0$ are from the hardware implementation aspect optimal only for a particular type of applications. Two types of algorithms exist: the first type proceeds by scanning the entire image and the second type propagates a narrow-band solution from the initial interface.

The algorithms using successive scannings are simple to implement. However, they suffer from a serious drawback that the next scan cannot begin before the previous one ends. Therefore, the scanning-based methods are optimal only for those applications where the solution on the entire image is required. A classical example is the algorithm of Danielsson (1980) which is based on two coordinates description and its overall complexity is $\mathscr{O}(N)$, where N is the number of points in the image. It yields the square of the distance function, which imposes to compute the square roots before the distance can be used for other computation, e.g. the curvature. Moreover, this algorithm is not conceived to operate with a sub-pixel precision. This problem is resolved in Tsai (2000) which proposes a *sweeping* algorithm (inspired from Boué and Dupuis (1999)). By using a new numerical scheme, it yields the exact distance and not the square and it reduces the numerical error of the classical Godunov scheme. The complexity of this algorithm is $\mathscr{O}(MN)$ where M is the number of data points and N is the number of grid points. However it is not possible to calculate the influence zones of different sources.

The algorithms operating in the narrow band are more appropriate for the propagation of labels. Their implementation is complicated by using sophisticated data structures and the complexity is $\mathscr{O}(Nlog(N))$. The Fast Marching introduced by Sethian (1996) is the most often used propagation technique in combination with the PDE-based methods. The algorithm allows to compute the distance function by realizing the principle of Huygens, as it is introduced in Verbeek and Verwer (1990), by propagating equidistant waves. For this, the Fast Marching algorithm needs to use an *ordered data structures* with a *real-number priority*. In every iteration can be processed only the point with the highest priority. The maximum priority represents a *global information* and makes this algorithm sequential. Moreover, the hardware implementation of data structures using a real-number priority is difficult because of operations like insertion, reading and re-positioning of elements.

This is not the case of the USP algorithm (Eggers, 1997) which does not use any sorted data structures. However, the result it yields is the square of the distance and it requires to memorize the current iteration number. It does not operate in sub-pixel accuracy and the complexity is $\mathscr{O}(n^3)$ for images of $n \times n$ points.

IMPLEMENTATION ISSUES

The filtering as well as the segmentation algorithms respond to some function of geometrical properties of the level-set function u such as gradient or curvature. These geometrical properties are obtained directly from the values of u or its derivatives (Sethian, 1996; Sapiro, 2000). The computation of the derivatives is an elementary operation. For every point concerned, these operations are performed on the nearest neighborhood and are independent each of the other. Hence they can be executed in parallel (Dejnožková, 2002).

Since the PDE-based algorithms principally consist in repetitive computation of the elementary

operations, the SIMD (Single Instruction Multiple Data Stream) architecture is the natural choice to reduce the processing time. We propose a divide-and-conquer approach in order to obtain a more balanced processors' activity and to limit the space on the chip. Thus one can benefit from a quasi parallel implementation by dividing the input data into blocks.

Recall that the SIMD architecture consists in an array of processors with an interconnection network for the communication. Each processor has its private non-shared memory. A single controller broadcasts instructions to all the processors. The processors then execute the instructions simultaneously at a given time.

The choice of the algorithm to implement and the architecture type come together. Compared to the filtering, other techniques as the continuous watershed or active contours require computation of a (weighted) distance to the given markers. These algorithms use sophisticated ordered data structures (such as hierarchical queues or a sorted heap) which can penalize the execution time on the SIMD architectures. The processing of such data structures introduces a sequential approach. Only one point (with maximal priority) can be processed at a time. Another reason why the SIMD architecture would be less efficient for this type of algorithms is the random access to the memory.

In the next section we show the implementation of the Massive Marching algorithm used for the computation of a distance function. This algorithm is fully parallel. Hence the execution time on an architecture with P processors is $t_{\text{parallel}} = t_{\text{sequential}}/P$.

MASSIVE MARCHING ALGORITHM

Throughout this paper we use the following notations. Let $p = [x_p, y_p]$ be a point of an isotropic, rectangular and unit grid. $V(p)$ denotes the 4-neighborhood of p defined as $V(p) = \{[x_p, y_p \pm 1], [x_p \pm 1, y_p]\}$. The point q is a neighbor of p if $q \in V(p)$, $u(p)$ denotes the value of the distance function in p.

NUMERICAL SCHEME

Numerical schemes allow to obtain the value of the distance function in a point according to the values of the neighbors. The numerical scheme is a discretization of the eikonal equation:

$$|\nabla u| = \mathscr{F}\,, \tag{2}$$

where $\mathscr{F}$ is the weight for a weighted distance. Some propositions of numerical schemes can be found in Sapiro (2000) or Tsai (2000). The most often used scheme is, in the domain of the Level Set, the scheme proposed by Godunov (Sethian, 1996).

$$\left[\max\left\{u(p) - u([x_p \pm 1, y_p]), 0\right\}^2 - \max\left\{u(p) - u([x_p, y_p \pm 1]), 0\right\}^2\right]^{\frac{1}{2}} = \mathscr{F}(p)\,. \tag{3}$$

In order to obtain the maximum values of the terms, we have to consider the neighbors with minimum values of u. The Godunov scheme requires to determine the maximal real solution of a quadratic equation (3).

Suppose that the distance u in the point p can be expressed by the following function of the neighborhood $V(p)$ of the point p and the weight $\mathscr{F}(p)$

$$u(p) = u_{\min}(p) + f_{\text{diff}}(V(p), \mathscr{F}(p))\,. \tag{4}$$

$u_{\min}(p)$ is the distance value of the minimal neighbor: $u_{\min}(p) = \min_{q_i \in V(p)}\{u(q_i)\}$. The formulation of f_{diff} depends on the choice of the numerical scheme. The Godunov scheme can be rewritten in the form of Eq. (4) where f_{diff} reads as

$$f_{\text{diff}} = \frac{|u_x(p) - u_y(p)|}{2} + \sqrt{\frac{\mathscr{F}(p)^2}{2} - \left(\frac{u_x(p) - u_y(p)}{2}\right)^2}\,, \tag{5}$$

and where $u_x(p) = \min\{u([x_p \pm 1, y_p])\}$ and $u_y(p) = \min\{u([x_p, y_p \pm 1])\}$

In Eq. (5) the minimum real solution is considered. If there is no real solution then the distance value is computed only from the minimal neighbor and $f_{\text{diff}} = \mathscr{F}(p)$.

INITIALIZATION

$\mathscr{C}_0$ is a closed curve which generally lies between the points of the grid $\mathbb{Z}^2$. Its accurate inter-pixel location is identified by the distance map u to $\mathscr{C}_0$. However, if it is to be described implicitly on a discrete support $\mathbb{Z}^2$, the curve $\mathscr{C}_0$ may not be placed arbitrarily. Its location is determined by the switching function $\varphi(p)$ where $\text{sign}(\varphi(p))$ indicates whether a given point p lies inside or outside $\mathscr{C}_0$ (as introduced in Sethian (1996)). Hence, $\mathscr{C}_0$ is located between adjacent points for which $\text{sign}(\varphi(.))$ differs. The exact distance u of these points to $\mathscr{C}_0$ is obtained by some interpolation method.

The choice of the interpolation depends on the requirements of the application. One can use either a constant value or a bilinear or a more sophisticated interpolation method allowing to detect more or

less complicated forms (for examples see Osher and Shu (1991); Siddiqi *et al.* (1997)). The majority of practical applications use a linear approximation. In the following, consider a linear interpolation and $|\varphi| = const.$ for all $p \in \mathbb{Z}^2$. Since u can only have a finite number of constant values for all points adjacent to $\mathscr{C}_0$, the initialization of the distance function reduces to $u : \mathbb{Z}^2 \to \{c_1, c_2, \ldots, c_n\}$, where all $c_i \in \mathbb{R}$. The number of the constants depends on the number of neighbors from which the interpolation is calculated. The Fig. 1 shows all the possible 4-neighborhood configurations for the linear interpolation.

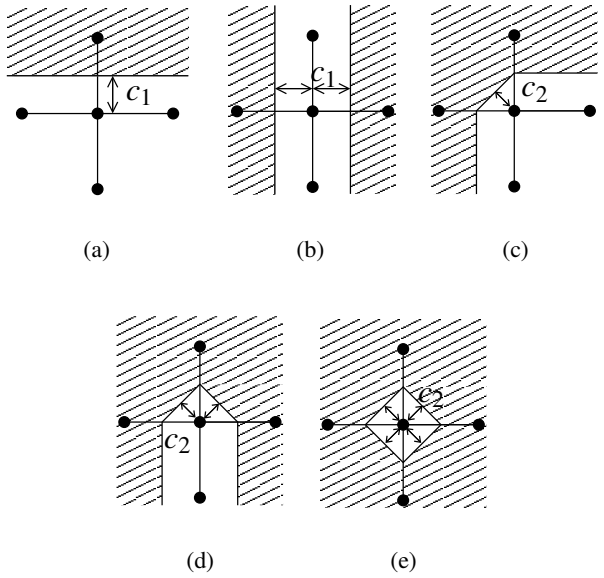


Fig. 1. *Configurations of the 4-neighborhood for the initialization.*

If the central point lies on the object's edge, *i.e.* the sign of neighbors changes with respect to the central (interpolated) point only in one direction (see Fig. 1(a) or 1(b)), the interpolation is computed only from $\varphi(.)$ of these neighbors. If the sign changes in both directions (corner or isolated point) the resulting value is computed from $\varphi(.)$ of the two neighbors (Fig. 1(c) to 1(e)).

Hence the initialization of $u(p)$ can be realized efficiently as a logical function of $\mathrm{sign}(\varphi(p))$, $\mathrm{sign}(\varphi(q_1))$, ..., $\mathrm{sign}(\varphi(q_4))$ and the result of the function is used to retrieve the corresponding value from a look-up-table containing the constants c_i.

PROPAGATION

We use the following sets to define the algorithm: $\mathscr{A}$ is the set of points initialized by the interpolation, $\mathscr{Q}$ is the set of points marked as active $\mathscr{Q} = \{q_i \mid q_i \notin \mathscr{A}$ and $V(q_i) \cap \mathscr{A} \neq \emptyset\}$. The algorithm reads as follows:

Initialization

- Initialize the neighborhood of the curve with a signed distance (set $\mathscr{A}$)
- Initialize the distance value u of the other points to ∞
- Mark the neighbors of $\mathscr{A}$ as *active* (set $\mathscr{Q}$)

Propagation

while $\mathscr{Q} \neq \{\}$, do in parallel for all $p \in \mathscr{Q}$:

{

⋆ *Compute new distance value*:

- *Jacobi step:*

$$u^{n+1}(p) = u^n_{\min}(p) + \min\{f_{\mathrm{diff}}(V(p), \mathscr{F}(p)), \mathscr{F}(p)\} \quad (6)$$

- *Gauss-Seidel step:*

$$u^{n+1}(p) = u^{n+1}_{\min}(p) + \min\{f_{\mathrm{diff}}(V(p), \mathscr{F}(p)), \mathscr{F}(p)\} \quad (7)$$

⋆ *Activation of new points to process:*

- delete p from $\mathscr{Q}$, insert p in $\mathscr{A}$
- if $u(p) < \mathrm{NB}_{\mathrm{width}}$ then for all q_i, $q_i \in V(p)$ such that $u^{n+1}(q_i) - u^{n+1}(p) > \varepsilon(q_i)$ (8)

  insert $q_i \to \mathscr{Q}$

}

where $\mathrm{NB}_{\mathrm{width}}$ is the desired width of the narrow band[1]. (The values of unprocessed points in t_n are automatically carried over to the next iteration and are noted as values at t_{n+1}.)

At each iteration, the value is calculated for the *active* points. The algorithm does not use any kind of sorted processing. Consequently, the front of the propagation is not equidistant to the initial curve. Two situations exist where the points that are currently being calculated will have to be reactivated later:

1. The value of the point is calculated on an incomplete neighborhood which imposes a *two-step algorithm*
2. The points are activated by a propagation front coming from a source which is not necessarily the closest one which is detected by *activation rule*.

[1] To obtain u on the entire image let $\mathrm{NB}_{\mathrm{width}} = \infty$

Two-step based algorithm

We say that the point value is computed on an incomplete neighborhood since adjacent (mutually dependent) points may be processed simultaneously. The calculation is therefore performed in two steps. The steps are named after their similarity with the algorithm of Markov chain approximation by PDE as introduced in Boué and Dupuis (1999). The first one, the *Jacobi step*, calculates the value of the distance function at t_{n+1} given only the values obtained at t_n. The second one, the *Gauss-Seidel step*, recalculates the distance value at t_{n+1} by using also the values obtained at t_{n+1}.

The algorithm computes the value $u(p)$ from the least neighbor. Hence, the infinite values are not considered for the computation. As mentioned above, in the Jacobi iteration, the first estimation of $u^{n+1}(p)$ is obtained by using the values of neighbors from the previous iteration which is recomputed once more in the Gauss-Seidel step.

Activation rule

The activation rule has two important roles: the supervision of the propagation end and the detection of the overlapping of the propagation waves (see Fig. 2).

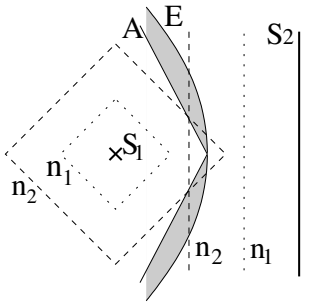


Fig. 2. *Example of an overlapping of the propagating waves and zone of reactivation (in gray). S_1, S_2 are the propagation sources. The dashed lines represent the propagation waves after n_1 and n_2 iterations.*

We calculate the distance function $u(x) = \min[\mathrm{dist}(x, S_1), \mathrm{dist}(x, S_2)]$. Let E denote the set of points equally distant from S_1 and S_2, $E = \{x | \mathrm{dist}(x, S_1) = \mathrm{dist}(x, S_1)\}$. The points to the left (resp. right) from E are closer to S_1 (resp. S_2). Note that in this example E is a parabola. A denotes the set of points activated simultaneously by the two fronts coming from the two sources. Since the propagation front of Massive Marching is not equidistant to the propagation source, the sets A and E do not coincide. The zone delimited by A and E contains points that were activated from S_2 whereas they are closer to S_1. These points will be reactivated again by the front coming from S_1 in order to lower their value from $\mathrm{dist}(x, S_2)$ to $\mathrm{dist}(x, S_1)$.

Suppose that $u^n(p)$ has just been calculated and p is deactivated. We search for an estimator of $u^{n+1}(q_i)$ to know whether the neighbor q_i of p should be activated in order to compute or to re-compute its value. Since the goal is to obtain the minimal solution the main idea is to test whether the value $u(q_i)$ could be brought down by considering p as the least neighbor of q_i. Suppose that p is the least neighbor of q_i. Then in the next iteration q_i will receive its value from p. From Eq. (4), $f_{\mathrm{diff}}(p)$ is the difference between the distance value $u(p)$ of a given point p and the least of the neighbors $u_{\min}(p)$. The new value $u(q_i)$ will satisfy $u(q_i) \geq u(p) + \inf f_{\mathrm{diff}}$.

Let $K_{\min}$ be the lower bound of f_{diff}:

$$K_{\min}(p) = \inf f_{\mathrm{diff}}(V(p), \mathscr{F}(p)) \,. \tag{9}$$

$\mathscr{F}(p)$ is an arbitrary but time invariant function. $K_{\min}$ is a predictor of the least increment of u in one iteration. All neighbors q_i of p such that $u(q_i) - u(p) > K_{\min}(p)$ should therefore be (re-)activated and (re-)calculated since the new value $u(p)$ may affect $u(q_i)$ in the next iteration. Hence, ε in Eq. (8) must satisfy:

$$\varepsilon(p) \geq K_{\min}(p) > 0 \,.$$

Remark: The lower bound of f_{diff} of the Godunov scheme (from the section 2.1) is

$$K_{\min}(p) = \sqrt{\frac{\mathscr{F}(p)^2}{2}} \,. \tag{10}$$

Note that $K_{\min}$ is constant whenever $\mathscr{F}$ is constant in (2) and becomes a function of $\mathscr{F}$ whenever $\mathscr{F}$ varies over the image.

Setting $\varepsilon < K_{\min}$ is useless because it would authorize the activation of points that will not be updated and the propagation could go backwards. By letting $\varepsilon > K_{\min}$ one can authorize fewer reactivations (lower execution time) paid by some error (proportional to $\varepsilon - K_{\min}$) in the result (Dejnožková, 2002).

LABEL PROPAGATION

The propagation of the region labels can be realized simultaneously with the computation of the distance function to obtain the influence zones for Voronoï tesselation or continuous watershed. Suppose that the region labels are initialized during the Massive Marching initialization stage. The algorithm modifies as the distance must be computed from neighbors having the same label.

- *Jacobi step*
  if $u_x(p)$ and $u_y(p)$ have the same label then use Eq. (6)
  else use $u^{n+1}(p) = u^n_{\min}(p) + F(p)$
- *Gauss-Seidel step*
  1. if $u_x(p)$ and u_y have the same label use Eq. (6)
     else use $u^{n+1}(p) = u^{n+1}_{\min}(p) + F(p)$
  2. p receives the label of $u^{n+1}(p)$

COMPUTATION ERROR

Two types of error exist: numerical scheme error and incomplete neighborhood error.

Numerical scheme error

As the majority of first order schemes also the Godunov scheme suffers from "shortsightedness" as it uses only the nearest neighborhood. Moreover, Tsai (2000) explains that when the propagation starts from isolated points it creates diamonds instead of circles. In Sethian (1999) Sethian proposes a switching mechanism between the first and second order scheme in order to improve the accuracy. Nevertheless, in ∞ the Godunov scheme converges to the exact solution.

Incomplete neighborhood error

This error is caused by the computation on an incomplete neighborhood (see section "Two-step based algorithm"). Massive Marching calculates simultaneously the values of adjacent points, *i.e.* values depending each on the others. Therefore the calculation is performed in two steps.

Also all the methods referenced in the introduction allow to recalculate the points several times in order to obtain more accurate solution to Eq. (2). The scanning-based methods recalculate during each scanning all the points of the image. Successive scannings have to be repeated until the convergence. Methods for the narrow band use a variable number of recalculations, implemented by using a sorted heap, depending locally on the neighborhood of every particular point. At every recalculation the point receives a new value of the distance according to the new values of the neighbors. Massive Marching authorizes to reactivate the neighbor q_i if the point p receives a new value $u(p)$ inferior to $u(q_i) - \varepsilon$ (see condition Eq. 8).

An additive error (typically at the fourth decimal place) may still appear in some special cases (as corners etc.), see Dejnožková (2002). The experiments have shown that the two-step calculation gives sufficient accuracy for most practical applications (see section "Experimental results"). Should more accurate results be required then the Gauss-Seidel step can be repeated.

COMPUTATION COMPLEXITY

In order to obtain the calculation complexity of Massive Marching we first assume that $\mathscr{F} = const.$ over the entire image.

The value of an active point p is obtained in a constant time $\mathscr{O}(1)$ after which the point deactivates itself. The point activates all its neighbors that verify the condition (8). The function (4) is strictly positive, no point can therefore reactivate the neighbor from which it has received the activation. If the propagation starts from one point representing the source, the algorithm complexity is $\mathscr{O}(N)$, with N be the number of points in the image.

For sources having more complicated geometrical forms or more than one source the complexity may exceed $\mathscr{O}(N)$ since some points may be activated more than once. We show that the number of reactivations is bounded. Consider two isolated points a and b such that there is a point c, $c \in V(a)$, $c \in V(b)$ and $a \notin V(b)$. Suppose that the two propagation fronts arrive respectively via a and b and meet in c. The two fronts have different speed and in the iteration n the distance values in a and b verify $|u^n(a) - u^n(b)| \geq 2K_{min}$. The faster front will stop in c whereas the slower one will continue. It can be shown that its propagation will stop after i iterations, where

$$i < \frac{u^n(b) - u^n(a)}{2K_{min}} - 1 \,.$$

In images with $\mathscr{F} = const.$, the waves propagate with unit increment of u in one iteration in vertical and horizontal direction, whereas in the diagonal directions the increment is obtained only after two iterations. See illustration at Fig. 2. The waves arriving from S_1 and S_2 meet first on the intersection of A and E since both waves have the same speed on the horizontal direction. Later, see the iteration n_2 for example, the waves meet outside the skeleton since the wave arriving from S_1 arrives diagonally and is therefore slower. The slower wave will continue its propagation up to the skeleton of the distance E where it stops.

For images with bounded support, the term $|u^n(a) - u^n(b)|$ is upper bounded and hence the number of reactivations also. For images where $\mathscr{F} \neq const.$, this term is also limited and depends on $\mathscr{F}$.

HARDWARE IMPLEMENTATION

The main implementation issues are outlined in the section "Implementation issues". In this section we present the implementation details and we discuss the possible extensions of the proposed architecture.

GLOBAL ARCHITECTURE

For the simulation and validation, we have chosen the division of the input image into the columns, *i.e.* one processing unit per column of the image. In a given moment the processing units process in parallel all the points in a row. We give below a description of the proposed (and tested) access to the neighbors. The processing units are controlled by the single Global Control block. It reads high-level instructions from the Code Memory. Each high level instruction indicates the action executed on the entire image in one scan and the code does not have to be decomposed in the elementary operations. The algorithm for computing a distance function on the entire image (given in section "Propagation") resumes to these high-level instructions:

interpolate
loop (if any point is active)
{ *Jacobi_Step*
Gauss-Seidel_Step }

The condition *any point is active* is the OR operation over all the activation flags.

The Global Control block not only controls the execution of the algorithm but also allows to change the values of the approximation registers inside each processing unit. Thus we can implement the approximation of almost any non-linear function.

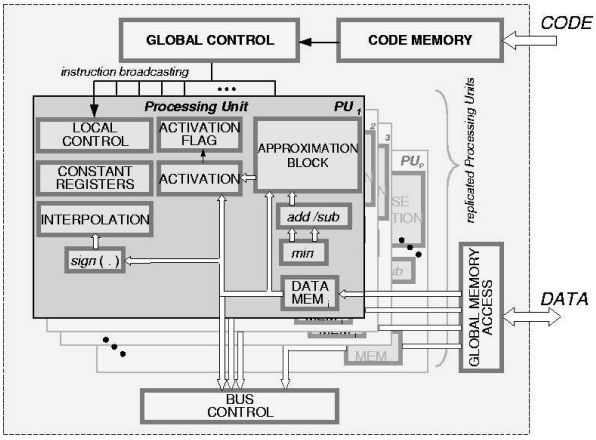


Fig. 3. *Global architecture with replicated Processing Units (PUs).*

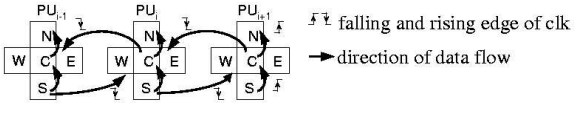
PROCESSING UNIT

The Processing Unit (see Fig. 3) contains specific blocks implementing different stages of the algorithm: the INTERPOLATION BLOCK ensures the initialization of the algorithm, and the APPROXIMATION BLOCK computes new pixel values. Each PU has a register containing the ACTIVATION FLAGs for its part of the image. The activation flag is used as a mask controlling the PU activity. All the Processing Units, whose currently processed point is active (the activation flag is set) execute the instructions, otherwise they are idle, except of sending their values to the east- and west-side neighbors.

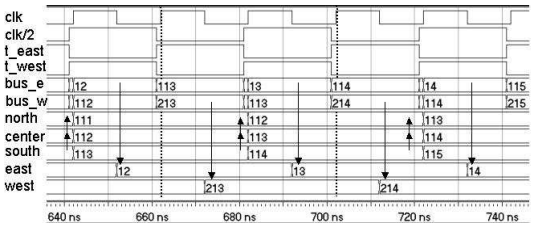
Note that the input data are stored in a non-shared data memory. Therefore, all the units can access to its data simultaneously at the given time. The memory is a double-port memory; before the execution of the algorithm, the data are uploaded by using a global access port (not given in the schematics) and read after. Recall that the Massive Marching uses the 4-neighborhood. In order to reduce the number of interconnections, we use bi-directional buses for the communication with the adjacent PUs. The bus direction is controlled by the signals t_east and t_west derived from $clk/2$.

Neighborhood retrieval

The complete neighborhood of a point (cf. Fig. 4(b)) is obtained in two clock cycles in the following way (Fig. 4(a)). With a rising edge of the clock a new SOUTH value is read from the local data memory whereas the old values SOUTH and CENTER are shifted upwards (e.g. 112, 113, 114). The values EAST and WEST are read from the bus on the falling clock edges. First, the WEST value is read from the west-side adjacent PU while the SOUTH point value is being sent to the east-side adjacent PU. On the next falling edge is read the EAST point while the CENTER point is being sent to the west-side adjacent PU. The complete neighborhood is ready immediately after the reading of the EAST point (indicated by the dashed line). The same communication protocol also applies to filtering.



(a)



(b)

Fig. 4. *Timing diagram of reading of the point neighborhood.*

Note that the choice of the image division affects only the communication procedure between the adjacent processing units and not their internal architecture.

Approximation block

Instead of computing the exact value of Eq. (5), requiring computation of a square and square root, we propose to use an approximation. The approximation allows to preserve the necessary sub-pixel precision while reducing the implementation cost. Two types of approximations have been tested, the piecewise linearization and look-up-table (LUT). The functional scheme of the tested approximation blocks is given by Fig. 5. The Eq. (5) is rewritten in the form

$$u(p) = \frac{u_x(p) + u_y(p)}{2} + g_{\text{diff}}\left(|u_x(p) - u_y(p)|\right) \quad (11)$$

which is, for the hardware implementation, approximated by either a linear approximation or a look-up-table

$$\hat{u}_{\text{Lin.Approx}}(p) = \frac{u_x(p) + u_y(p)}{2} + a_i(|u_x - u_y|) + b_i \quad (12)$$

$$\hat{u}_{\text{LUT}}(p) = \frac{u_x(p) + u_y(p)}{2} + a_i \quad (13)$$

The number of operations to obtain $\hat{u}_{\text{Lin.Approx}}(p)$ reduces to three additions, one subtraction and two multiplications and for $\hat{u}_{\text{LUT}}(p)$ only two additions and one multiplication (paid by higher memory requirements for comparable accuracy). The computation is done with a fixed-point precision. A comparative study of the implementation cost is presented below.

The implementation of the approximation is given by Fig. 5. The input signals are $|u_x - u_y|$ and $u_x + u_y$. (The terms u_x and u_y are obtained by two comparators in the MIN block (Fig. 3)). The former enters also in the Interval Search block generating the address (interval number) of the register containing the corresponding approximation constants a_i and b_i (cf. Fig. 6(a)). The SELECTION CONTROL block is testing whether the values u_x, u_y are finite. If both values are finite (pixels have already been activated) then the distance is computed by using Eq. (11). If only one of the values u_x, u_y is finite then the distance computation reduces to addition of one to the finite value: the approximation constants are replaced in order to add 1 to the minimal neighbor. Recall that both values cannot be infinite in the same time since such a point would not be activated.

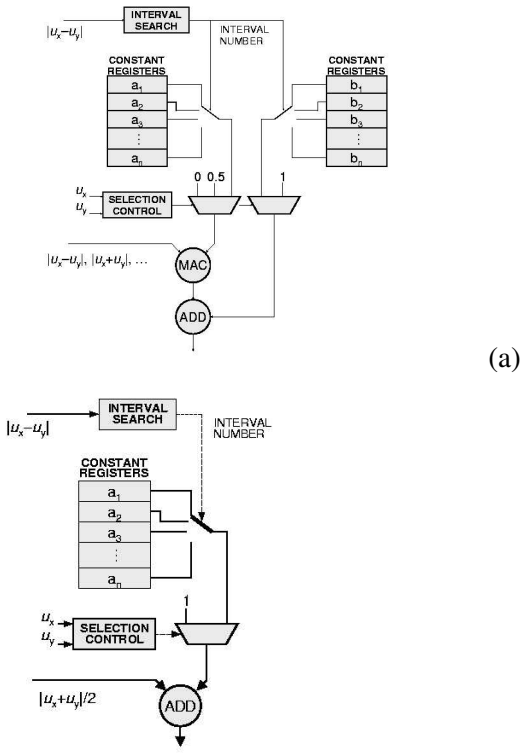


Fig. 5. *Internal block architecture of the Approximation Block; (a) Piecewise linearization, (b) Look-Up-Table.*

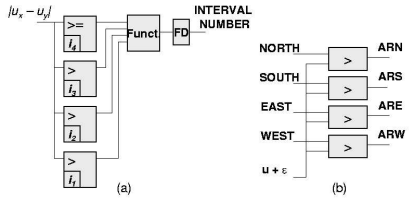


Fig. 6. *Interval Search and Activation Request block.*

The computation process is completely pipelined. After an initial latency of 10 clock periods, the result is obtained in one clock period. The approximation block as it is given here can calculate the distance in one clock cycle only for $\mathscr{F} = 1$. For $\mathscr{F} \neq 1$, the multiplier must perform two additional multiplications. The overall bandwidth of the architecture will be lower unless two additional multipliers (or approximations) are used.

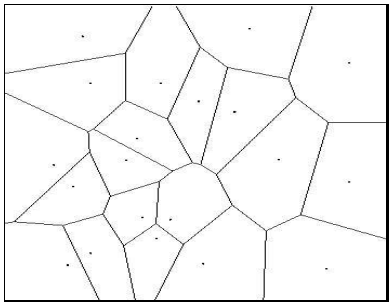
Pixel activation

Each pixel has its own flag controlling the activity status of the processing unit. It indicates whether the new value is to be computed or not. The activation flag of the point x gets active whenever the condition Eq. (8) is verified. The active points are testing their activity for the next iteration by using the condition Eq. (8) and may also activate their inactive neighbors by sending them an activation request (see Fig. 6(b) for

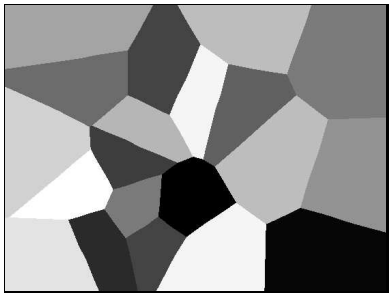
signals ARN, ARS, ARE, ARW - Activation Request to North, South, etc.). As soon as all the flags are inactive the algorithm ends.

EXPERIMENTAL RESULTS

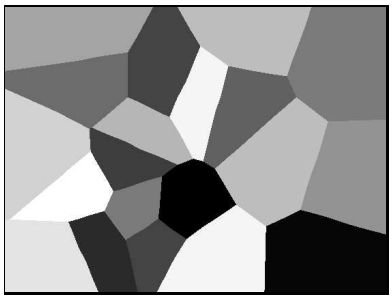
In order to prove the validity of the algorithm, we illustrate the behaviour of Massive Marching on computation of the Voronoï tesselation for a given set of points in a 2D euclidian space. In this case, we consider $\mathscr{F}(p) = 1$ for $\forall p \in \mathscr{P}$ (see Fig. 7). Note that if needed, the propagation of labels can be done simultaneously with the propagation of the distance.



(a) Exact, computed on Delaunay triangulation



(b) Massive Marching

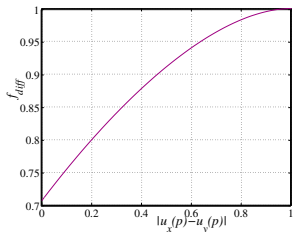


(c) Fast Marching

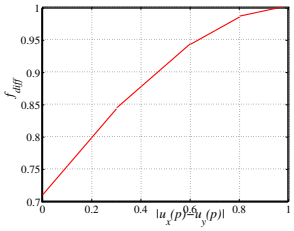
Fig. 7. *The Voronoï tessellation obtained by Massive Marching, compared to Fast Marching.*

The result achieved by Massive Marching is compared to the exact result and to the result obtained by the sorted heap algorithm (Sethian, 1996). Slight differerence is due to i) an error of the Fast Marching induced by the direction of the scanning and ii) an approximation error of Massive Marching.

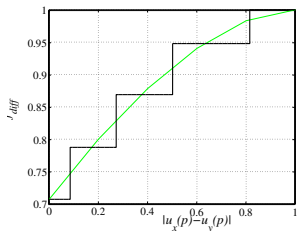
We have tested the accuracy of the result with respect to two factors: i) the approximation type of the Godunov numerical scheme and ii) the number of fractional bits of the fixed-point implementation of the approximation block. We have observed the error in ℓ_∞ in the result with respect to the exact solution simulated with "double" precision in C. Recall that the norm ℓ_∞ corresponds to the maximum of the vector elements. Here, it represents the upper bound of the error.



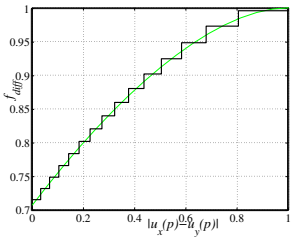
(a) Original function of the Godunov scheme



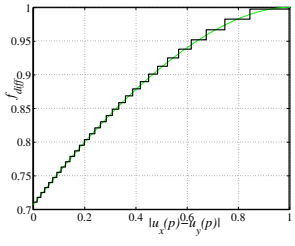
(b) Linear approximation with 4 intervals



(c) Inequally spaced LUT approximation with 5 intervals



(d) Inequally spaced LUT approximation with 15 intervals



(e) Inequally spaced LUT approximation with 30 intervals

Fig. 8. *Different approximations of* f_{diff}.

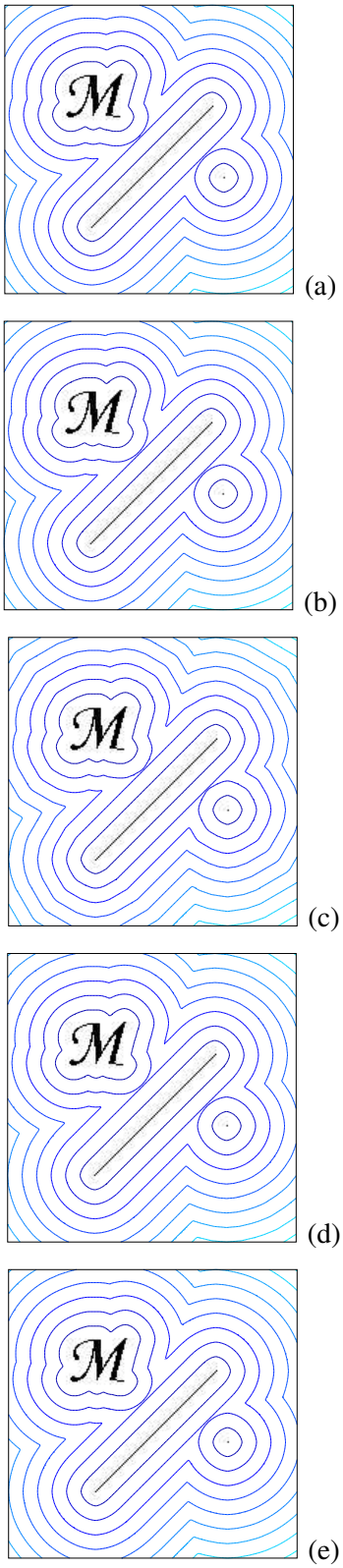


Fig. 9. *Iso-distance lines obtained with various approximations (8+8 bits of precision); (a) Original function of the Godunov scheme, (b) Linear approximation with 4 intervals, (c) LUT approximation with 5 intervals, (d) LUT approximation with 15 intervals, (e) LUT approximation with 30 intervals.*

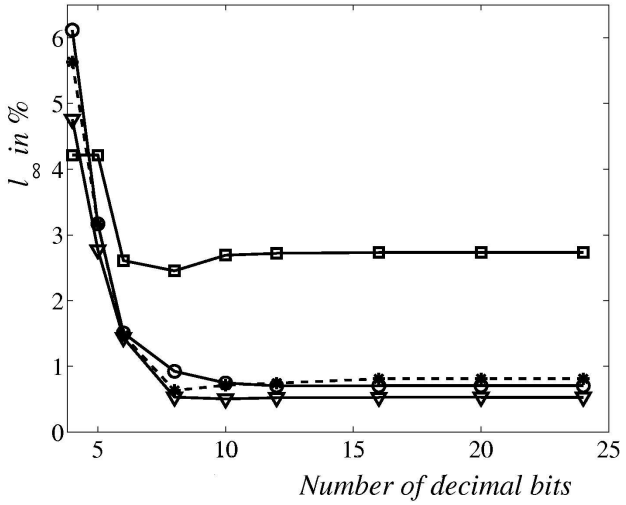


Fig. 10. *Error (in ℓ_∞) of the look-up-table approximation compared to the exact solution depending on the fractional part width. Rectangles: LUT 5 intervals, circles: LUT 15 intervals, triangles: LUT 30 intervals; asterisks: piecewise linearization.*

Two approximation types of the function g_{diff} were used: a *piecewise linearization* with four intervals and *look-up-table* approximation with five, fifteen and thirty steps, see Fig. 8. The first and the last elements in the look-up-tables are exact (equal to $1/\sqrt{2}$ and 1) in order to minimize the error in the left, right, up and down and diagonal directions. The other values are obtained as to distribute the error evenly over the entire interval $]0, 1[$. The distance results can be visually assessed in Fig. 9 on the iso-distance lines given in the same figure for 8+8 bit precision. The test image contains three sources: a letter M, straight line and a point. The approximation error (in ℓ_∞) with respect to the exact result is given in Table 1.

Table 1. *Error of approximation of the Godunov scheme given by Fig. 9.*

Approx. type (interval no.)	Linear. 4	LUT 5	LUT 15	LUT 30
Error ℓ_∞ (%)	0.3	3.9	1.2	0.9

The implementation of the approximation block (cf. Fig. 5) was tested in fixed-point precision with 8 bits for the integer part and 4, 6, 8, 10, 12, 16, 20 and 24 bits for the fractional part. Note that the eight-bit integer part limits the distance to 0 to 255 and has to be increased if needed more. The overall (approximation plus rounding) error is given by Table 2.

Table 2. *Overall (approximation plus rounding) error in ℓ_∞ of the result in Fig. 9(b, c, d and e) to the exact result Fig. 9(a).*

Approx. type (interval no.)	Linear. 4	LUT 5	LUT 15	LUT 30
Error ℓ_∞ 8 bits (%)	0.64	2.45	0.92	0.54
Error ℓ_∞ 16 bits (%)	0.81	2.74	0.70	0.53

Simultaneously with the error introduced by the approximation and rounding error, see Fig. 10, we have observed the implementation cost in terms of the number of equivalent NAND gates in the netlist, reported by the compilator, before the optimization and routing on a specific chip (only for precision 8+8 and 8+16 bit integer+fractional part). For the surface estimation cf. Tables 3 and 4 below.

Table 3. *Surface estimation of the approximation block 8+8 bits of precision (integer+fractional part).*

Approx. type (interval no.)	Piecewise lin. 4	LUT 5	LUT 15	LUT 30
Surface after optimization (NANDs)	10132	4031	6920	13856
Memory bits	128	80	240	480

Table 4. *Surface estimation of the approximation block 8+16 bits of precision (integer+fractional part).*

Approx. type (interval no.)	Piecewise lin. 4	LUT 5	LUT 15	LUT 30
Surface after optimization (NANDs)	20044	5743	9056	16592
Memory bits	192	120	360	720

Note that when using the piecewise linearization the surface requirements increase considerably (about twice of equivalent NANDs) if the accuracy increases from 8+8 to 8+16 bits (integer plus fractional part). On the other hand the surface occupation grows linearly when the look-up-table is used: increase by some 20% to 40% of equivalent NANDs and by one third of memory bits. The nonlinear increase of the implementation cost between the 8+8 and the 8+16 accuracy is due to the use of a highly optimized multiplier/accumulator.

CONCLUSIONS

This paper proposes a SIMD-type architecture for curve-evolution PDEs. This architecture has already been used for filtering by linear diffusion, see Gijbels *et al.* (1994). In this paper is shown that the same architecture type can also be used for the narrow-band like algorithms.

Obviously, the activity of Processing Units arranged in a linear array is unbalanced for narrow-band like algorithms. The activity distribution depends on the geometric form of the objects. This inconvenience is the price paid for the advantage that without major modifications this architecture can run algorithms consisting of several stages, e.g. filtering followed by watershed or voronoï tessellation computation. The only modification consists in reconfiguration of the approximation blocks by uploading new values in the look-up-tables or the linear approximation registers. The algorithm is then executed by broadcasting corresponding high level instructions to the Processing Units.

For implementation on a FPGA, the maximum clock frequency we have obtained is 150MHz. One point is processed in two clock cycles (Jacobi plus Gauss-Seidel) which gives a theoretical bandwidth of one Processing Unit 75×10^6 points^{-1}. The worst execution time estimation for the QCIF format (176 pixels wide by 144 high) with the distance source in a corner is 400 µs.

We have observed that even if implemented sequentially in some situations (for denoised filtered images) the Massive Marching outperforms algorithms using ordered structures. For heavily noised input images, the Massive Marching performance remains comparable to other algorithms despite frequent reactivations.

Future work: Extension of Massive Marching to 3D seems promising. The execution of other

algorithms on big 3D images is penalized by excessive memory requirements of large, real-number-priority ordered structures. The use of Massive Marching may be advantageous because of the absence of any ordered waiting structures.

A better activity distribution would be achieved with processing units retrieving points waiting in a FIFO-like queue. Suppose a completely pipelined, random-access capable prefetch so that the neighborhood is retrieved in one clock cycle. The theoretical execution time will be $\mathcal{O}(N)/P$ cycles, where P is the number of pipelined processing units. The surface activity will be more balanced. An efficient neighborhood prefetch therefore needs to be found so that several pipelined processing units could be used.

REFERENCES

Boué M, Dupuis P (1999). Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control. SIAM J Numer Anal 36:667–95.

Danielsson P (1980). Euclidean distance mapping. Comput Vision Graph 14:227–48.

Dejnožková E (2002). Massive marching : A parallel computation of distance function for PDE-based applications. Tech. Rep. N-17/02/MM, ENSMP, Center of Mathematical Morphology.

Dejnožková E, Dokládal P (2003a). A multiprocessor architecture for PDE-based applications. Visual Information Engineering, VIE 2003. Proceedings.

Dejnožková E, Dokládal P (2003b). A parallel algorithm for solving eikonal equation. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP. Proceedings.

Eggers H (1997). Fast parallel euclidian distance transformation in $\mathbf{Z^n}$. SPIE Proceedings 3168:33–40.

Gijbels T, Six P, Gool LV, Catthoor F, Man HD, Oosterlinck A (1994). A VLSI architecture for parallel non-linear diffusion with applications in vision. IEEE Workshop on VLSI Signal Processing .

Gomez J, Faugeras O (1992). Reconciling distance functions and level sets. Tech Rep No: 3666, INRIA.

Kimmel R (1995). Curve Evolution on Surfaces. Ph.D. thesis, Technion Israel Institute of Technology.

Kimmel R, Sethian JA (2001). Optimal algorithm for shape from shading and path planning. J Math Imaging Vis 14:237–44.

Meyer F, Maragos P (1999). Multiscale morphological segmentations based on watershed, flooding, and eikonal PDE. In: Nielsen M, Johansen P, Olsen O, Weickert J, eds., Scale-Space Theories in Computer Vision, no. 1682 in Lecture Notes in Computer Science. Springer-Verlag, 351–62.

Osher S, Sethian J (1988). Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. J Comput Phys 79:12–49.

Osher S, Shu CW (1991). High-order Essentially Non-oscillatory schemes for Hamilton-Jacobi equations. SIAM J Numer Anal 28:907–22.

Paragios N (2000). Geodesic Active Regions and Level Set Methods : Contributions and Applications in Artificial Vision. Ph.D. thesis, Université de Sophia Antipolis.

Perona P, Malik J (1988). A network for multiscale segmentation. Proceedings IEEE International Symposium Circuits and Systems CISCAC88 :2565–8.

Rumpf M, Strzodka R (2001). Level set segmentation in graphics hardware. In: Proceedings ICIP 2001.

Sapiro G (2000). Geometric Partial Differential Equations and Image Analysis. Cambridge: Cambridge University Press.

Sethian JA (1996). Level Set Methods. Cambridge: Cambridge University Press.

Sethian JA (1999). Fast marching methods. SIAM Review 41:199–235.

Siddiqi K, Kimia B, Shu CW (1997). Geometric shock-capturing ENO schemes for subpixel interpolation, computation and curve evolution. Graph Model Im Proc 59:278–301.

Suri J, Singh S, Reden L (2002). Computer vision and pattern recognition techniques for 2-D and 3-D MR cerebral cortical segmentation: A state-of-the-art review. Int J Patt Anal Ap 5:46–76.

Tsai Y (2000). Rapid and accurate computation of the distance function using grids. Tech. Rep. 17, Department of Mathematics, University of California, Los Angeles.

Verbeek P, Verwer B (1990). Shading from shape, the eikonal equation solved by grayweighted distance transform. Patt Recogn Lett 11:681–90.

Weickert J, ter Haar Romeny BM, Viergver MA (1998). Efficient and reliable schemes for nonlinear diffusion filtering. In: IEEE Trans Image Proc, vol. 7. 398–410.

Zhao H, Chan T, Merriman B, Osher S (1996). A variational level set approach to multiphase motion. J Comput Phys 127:179–95.

A preliminary (accepted) version of an original paper to appear in **Journal of Real-Time Image Processing**
(The final publication is available at Springer via http://dx.doi.org/10.1007/s11554-015-0518-2)

Jan Bartovský · Petr Dokládal · Matthieu Faessel · Eva Dokládalová · Michel Bilodeau

Morphological Co-Processing Unit for Embedded Devices

June 25, 2015

Abstract This paper focuses on the development of a fully programmable morphological coprocessor for embedded devices. It is a well-known fact that the majority of morphological processing operations are composed of a (potentially large) number of sequential elementary operators. At the same time, the industrial context induces a high demand on robustness and decision liability that makes the application even more demanding. Recent stationary platforms (PC, GPU, clusters) no more represent a computational bottleneck in real-time vision or image processing applications. However, in embedded solutions such applications still hit computational limits.

The Morphological Co-Processing Unit (MCPU) replies to this demand. It assembles the previously published efficient dilation/erosion units with geodesic units and ALUs to support a larger collection of morphological operations, from a simple dilation to a serial filters involving a geodesic reconstruction step.

The coprocessor has been integrated into an FPGA platform running a server, able to respond client's requests over the ethernet. The experimental performance of the MCPU measured on a wide set of operations brings as results in orders of magnitude better than another embedded platform, built around an ARM A9 quad-core processor.

J. Bartovský
Centre for Mathematical Morphology, MINES ParisTech, Fontainebleau, France.
Faculty of Electrical Engineering, University of West Bohemia, Pilsen, Czech Republic.
E-mail: jan.bartovsky@mines-paristech.fr

P. Dokládal, M. Faessel and M. Bilodeau
Centre for Mathematical Morphology, MINES ParisTech, Fontainebleau, France.
E-mail: {matthieu.faessel, petr.dokladal, michel.bilodeau}@mines-paritech.fr

E. Dokládalová
Computer Science Laboratory Gaspard Monge, ESIEE Paris, University Paris-Est, Noisy-le-Grand, France.
E-mail: e.dokladalova@esiee.fr

Keywords Mathematical Morphology, Hardware Implementation, Pattern Spectrum, Reconstruction, Parallel Computation

1 Introduction

Mathematical morphology is an image processing framework providing a complete set of tools for filtering, multiscale image analysis, or pattern recognition. It is used in a number of applications, including biomedical and medical imaging, video surveillance, industrial control, video compression, stereology or remote sensing since its very first appearance in the late 1960's, see [21, 27–29].

Considering the hardware implementation context, several different trends have been observed. A recent technological advance of imaging sensors stimulated the development of applications by means of high-resolution images that became a standard. Needless to say large images impose challenging requirements on the computation platform in terms of both performance and memory.

On the other hand, the industrial context often induces severe real-time constraints on applications. Often these demanding image-interpretation applications require a high correct-decision liability, robust but costly multi-criteria and/or multi-scale analyses are used. Given that image processing should not deteriorate industrial productivity, the latency and computational performance are of high interest in this context.

In embedded systems, the most important concerns are low power consumption (and consequently low heat dissipation) and small resources occupation, which allows for better embedding. All these considerations combined together infer overwhelming requirements on the architecture of polyvalent processing units addressing many different contexts. The context of embedded morphology applications includes, for instance, an augmented vision system that improves visual perception [12], or smart cameras [15].

This paper stems from a previous work [1, 2, 9] and shows how to build around the computation pipelines an advanced, polyvalent and user-friendly computation platform

for embedded or portable applications. We have enriched the set of supported operators by adding geodesic units, ALUs, a fast DDR2 memory controller and a fast ethernet link to transfer images. The computation is controlled and monitored by a Xilinx MicroBlaze microcontroller that also ensures the communication with the outside world. The entire platform behaves as a server and responds to client's computation requests. A provided software interface (in python and C/C++) offers the user a convenient possibility to interact with the platform.

The text is organized as follows: Section 2 makes a short survey of existing morphological algorithms and architectures. Section 3 outlines the basic definitions of typical mathematical morphology operations. Section 4 describes how these operations can be efficiently computed by processing pipelines and describes the architecture of the proposed coprocessor. The following Section 5 covers the programmability and user interface to the coprocessor server. Finally, Section 8 presents experimental results obtained on an FPGA board and compares them to an ARM A9 embedded platform.

2 State of the art

This section briefly presents the state of the art algorithms for elementary morphology operations dilation and erosion and their hardware implementations in FPGA. The last part discusses the novelty and main contributions of this paper.

2.1 Algorithms

The simplest method to compute a dilation is the exhaustive search for maximum in the scope of a structuring element (SE) B according to the definition Eq. 1 in Section 3. This naive solution tends to need a large number of comparisons, which are on most platforms diadic (with two operands). The number of comparisons is considered as a metric of algorithm complexity, so the naive algorithm has complexity $\mathcal{O}(l)$ as it has to carry out $l-1$ comparisons for a SE containing l pixels. Such complexity suggests that the naive algorithm is inefficient for large SEs. Pecht [25] proposed a method to decrease the complexity based on the logarithmic SE decomposition, thereby achieving $\mathcal{O}(\lceil \log_2(l) \rceil)$ complexity.

The first 1-D algorithm that reduced the complexity to a constant is by van Herk [34], and Gil and Werman [14] (published simultaneously in two papers and often referred to by the authors' initials as HGW). The computation complexity is constant, i.e., of $\mathcal{O}(1)$, which means the upper bound of the computation time is independent of the SE size. Gil [13] proposed an improved version of HGW that lowered the number of comparisons per element, but at the cost of increased memory usage and implementation complexity.

Lemire [18] proposed a fast stream algorithm of $\mathcal{O}(1)$ for causal line SEs. This algorithm uses two queues of length l in order to store the pixels that form locally monotonous signal (i.e., monotonously increasing and decreasing). Although it produces both erosion and dilation simultaneously, it works with causal SEs only. This downside was solved later by Dokládal [9] who proposed another queue-based algorithm. The advantages of these queue-based algorithms are strictly sequential access to data, zero latency, and low memory requirements.

The 2-D dilation is usually obtained by composition of 1-D dilations, see for instance Soille [31] who approximates circle and polygon SEs using rotated line SEs. However, this technique covers only a limited family of shapes. The arbitrary-shaped SE are obtained by either more complex 2-D algorithms (e.g., Urbach [33]), which are suitable for general-purpose processors, or by fine-grained decomposition of the large SE into a set of small 2-D SEs. Xu [41] proposed that any 8-convex polygon (convex on 8-connectivity grid, hence 8-convex) is decomposable into a class of 13 nontrivial indecomposable convex polygonal SEs. Normand [23] reduced the class of shapes to only four 2-pixel SEs by allowing the union operator to take place in the SE decomposition.

2.2 Hardware implementations

One of the first morphology architectures was the texture analyzer by Klein [17]. It was optimized for linear and rectangular SE by decomposition into line segments. More recently, Velten [35] proposed another, delay-line based architecture for binary images supporting arbitrarily shaped 3×3 SEs. The computation of dilation is realized by OR gates (topology was not communicated, probably a tree of diadic OR gates) achieving good performance, which was further improved by spatial parallelism.

Clienti [4] proposed a highly parallel morphological System-on-Chip. It is a set of neighborhood processors optimized for arbitrarily shaped 3×3 SE interconnected in a partially configurable pipeline. Each stage of the pipeline contains 2 processors that can process 2 parallel image streams and an ALU. The reconfiguration allows all the processors to be connected in one chain in order to employ all processors when only one image stream is used. A reconfigurable 3x3 neighborhood morpho processor was recently used in Gibson [12] in a hand-held augmented-vision system for visually impaired.

Another approach is called partial-result reuse (PRR). The morphological operation by some neighborhood B_1 in an early stage is delayed by delay lines in order to be reused later in computation by some other neighborhood B_2 obtaining larger B_3 decreasing thus the number of necessary comparisons. One of the first PRR architectures for 1-D dilation was proposed in [26] and improved in [6]. The principle is based on an exponential growth of the intermediate neighborhoods in the partial-result reuse scheme.

Chien [3] presented more general concept of PRR that builds the desired SE by a set of distinct partial neighborhoods computed by a dedicated algorithm. As a result, it

supports arbitrary 8-convex polygon at the cost of some additional comparisons.

A similar approach is based on the Normand [23] decomposition where a convex SE is decomposed into a number of causal 2-pixel SEs, applied in sequence or in parallel. Déforges *et al.* [8] make use of this decomposition, which combined with a stream implementation, allows to conceive a pipeline architecture supporting arbitrary convex SEs.

Recently, Torres-Huitzil [32] designed a linear systolic-like array of processing elements without need for delay-line internal memory storage supporting non-rectangular flat SEs. However, prospective drawbacks can be seen in the chosen column-based image scan requiring significant image storage capability, and the need of deep parallelism to attain real-time performance even for the mentioned 7×7 SE.

The last method mentioned in this overview is the implementation of efficient 1-D algorithms. To our knowledge, there are only few such contributions in the literature. Clienti [5] published architectures for the 1-D Lemonier algorithm [19] and the HGW algorithm [14,34]. Both algorithms require a reverse scan which increases the memory requirements. Regarding the HGW algorithm the line can be reversed per blocks [5], which significantly decreases the latency.

Recently, Bartovský [1] proposed an implementation of the Dokládal algorithm [9] as a processing unit by polygonal SEs with a strictly sequential access to data and small memory requirements. This architecture is mainly beneficial for large SEs because only the memory varies with the size of the SE, the computation logic remains the same.

3 Basic notions

This section describes the operators used in this paper. We are mainly interested in compound operators composed as concatenations of elementary operators, that are therefore costly on sequential machines.

Let δ_B, ε_B: $\mathbb{Z}^2 \to \mathbb{R}$ be a dilation and an erosion on gray-scale images, parameterized by a structuring element B, assumed to be flat (i.e., $B \subset \mathbb{Z}^2$) and translation-invariant, defined as [27,30]

$$\delta_B(f) = \bigvee_{b\in B} f_b \; ; \qquad \varepsilon_B(f) = \bigwedge_{b\in \widehat{B}} f_b \tag{1}$$

where f_b denotes translation of f by b. The hat $\widehat{\ }$ denotes the transposition of B, equal to the set reflection $\widehat{B} = \{x \mid -x \in B\}$.

For an image f, its internal, external and morphological gradients are given by

$$g_i(f) = f - \varepsilon_B(f) \tag{2}$$
$$g_e(f) = \delta_B(f) - f \tag{3}$$
$$g(f) = \delta_B(f) - \varepsilon_B(f) \tag{4}$$

The concatenation of dilation and erosion forms other morphological operators. The closing and opening on gray-scale images, φ_B, γ_B: $\mathbb{Z}^2 \to \mathbb{R}$, parameterized by a structuring element B, are defined as

$$\varphi_B(f) = \varepsilon_B[\delta_B(f)] \; ; \qquad \gamma_B(f) = \delta_B[\varepsilon_B(f)] \tag{5}$$

The residue of an opening or closing is called top-hat,

$$th^{\gamma}(f) = f - \gamma_B(f) \; ; \qquad th^{\varphi}(f) = \varphi_B(f) - f \tag{6}$$

and is used to detect objects filtered by the opening or the closing.

Closing and opening are filters. Their concatenation forms alternating filters $\gamma\varphi$, $\varphi\gamma$, $\gamma\varphi\gamma$ and $\varphi\gamma\varphi$. Other filters can be obtained by combining *families* of filters. A well known example is the alternating sequential filter (ASF), composed as sequence of closings and openings with a progressively increasing SE λB, with $\lambda > 0$. Let γ^{λ} and φ^{λ} denote the change of scale such as $\gamma_{\lambda B}$ and $\varphi_{\lambda B}$. Then λ-order ASF (referred to as ASF$^{\lambda}$) is composed as

$$\text{ASF}^{\lambda} = \varphi^{\lambda}\gamma^{\lambda}\varphi^{\lambda-1}\gamma^{\lambda-1}\ldots\varphi^{1}\gamma^{1} \tag{7}$$

starting with opening, and

$$\text{ASF}^{\lambda} = \gamma^{\lambda}\varphi^{\lambda}\gamma^{\lambda-1}\varphi^{\lambda-1}\ldots\gamma^{1}\varphi^{1} \tag{8}$$

starting with closing.

Let $\delta_f : \mathbb{Z}^2 \to \mathbb{R}$ be an elementary geodesic dilation of image g (marker) "under" image f (mask) where $g \leq f$, such as [36]

$$\delta_f(g) = f \wedge \delta_{3\times3}(g) \tag{9}$$

Repeating $\delta_f(f)$ until stability represents the dilation-reconstruction of g under f; $g \leq f$,

$$\rho_f(g) = \underbrace{\delta_f\delta_f\ldots\delta_f}_{x \text{ times}}(g) \tag{10}$$

the number of iterations $x = \infty$ by definition, and practically until the idempotence. The marker image g is commonly obtained by morphological opening γ_B. In this case, the operation is called opening by reconstruction γ^{ρ}_B, defined as

$$\gamma^{\rho}_B(f) = \rho_f(\gamma_B(f)) \tag{11}$$

Let $\{\gamma^{\lambda_i}\}$, with $\lambda_i > \lambda_{i-1}$ and with $\lambda_i > 0$, $\forall i$ be a collection of openings, generating a size distribution aka granulometric function (see Matherons' axioms, [21] p. 192) using some measure, e.g. integral (or sum) of the image. One also often uses its derivative, so called granulometric or pattern spectrum, defined as

$$PS_{\lambda_j B}(f) = \sum_{D} \left(\gamma_{\lambda_i B} f - \gamma_{\lambda_j B} f\right) \tag{12}$$

with $D = \text{spt}(f)$. Notice, that instead of opening $\gamma_{\lambda_i B}$ one also may want to use the opening by reconstruction $\gamma^{\rho}_{\lambda_i B}$ which even more increases the computation cost.

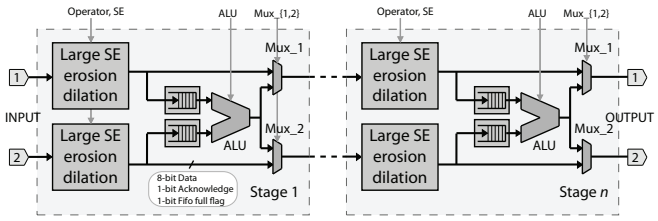


Fig. 1 Large SE pipeline architecture.

4 Hardware architecture

This section describes the hardware architecture of the proposed morphological coprocessor that efficiently implements the aforementioned sequential, costly operators. The following description follows the bottom-up approach, so we start with developing two basic image processing pipelines, one for large SE operators, and one for geodesic operators. Then we build the processing core by surrounding these two pipelines with interconnection busses, configuration registers, and image buffers in such way that it can be used as a peripheral of the Xilinx MicroBlaze. Finally, we describe the top-level architecture of the FPGA evaluation platform.

4.1 Large SE Pipeline

Let us begin with the description of the Large SE pipeline.

One of the most penalizing aspects in morphological operators is the number of iteration on an image. On sequential platforms, this induces an intensive traffic between the CPU and the memory. A considerable increase in efficiency can be obtained when such iterations can be pipelined. This idea is used in Clienti [5]. However, given that its pipeline is only composed of 3×3 blocks it lacks flexibility and must be reconfigured to precisely fit application needs.

In this paper, in order to gain in flexibility we propose two parallel pipelines of programmable large-size SE units. To efficiently execute the largest collection of operators (including those with two parallel branches, such as top hat or gradient) the pipelines are interconnected by programmable ALUs (Arithmetic Logic Unit), see Fig. 1.

Both pipelines contain several identical parts called processing stages connected one after the other. The pipeline is scalable by means of the number of instantiated stages, which is hereafter denoted as n. The heart of each stage is a pair of Large SE erosion/dilation units. The output of both units can be connected to the ALU, or directly to the next stage through the Mux_{1,2} multiplexer. The ALU result can be routed to either input port of the next stage (or both). This stream routing capability increases the adaptivity of the architecture to execute different algorithms. See below (Fig. 4), a few examples of possible interconnections.

The Large SE erosion/dilation unit performs one morphological dilation or erosion by a flat rectangular or octagonal SE of programmable size (up to 31 pixels in diameter for rectangles and 43 pixels in diameter for octagons) and the position of the origin. This unit takes advantage of separability of 2-D rectangular and octagonal SEs into a sequence of 1-D SEs. The separability allows that the computation can be separated into rows and columns for rectangles, and four obliquous (rotated by 45^o) lines for octagons. This simplifies the memory manegement since the data can be stored in small-size independent memory blocks. These blocks behave as queues if the 1-D dilation is computed by a queue-based algorithm [9]. A maximum allowable size of the SE needs to be specified prior to the synthesis. Afterwards, the size of the SE is freely programmable by the user within the range of the synthesize maximum size. In the same way, the size of the image is programmable withing the maximum specified prior to the synthesis. Such implementation has proven to be flexible, and beneficial for high-demanding image processing with large SEs. The description of the FPGA architecture and experimental results can be found in [2] for rectangular SEs, and [1] for octagonal SEs, respectively.

The previously published results can be summarized as follows. The Large SE unit computes 2-D rectangular or octagonal erosion/dilation during a single horizontal image scan with minimal latency. The experimentally obtained average processing rate is approximately 2.5 clock cycles per pixel, i.e., approx. 50 Mpx/s at 125 MHz clock frequency. The memory requirement is another important parameter of an image processing implementation because it limits the number of units that fit in the FPGA. The most significant memory requirement of the Large SE unit is given by the set of queues, such as

$$R = NH \times (bpp + \lceil \log_2(H-1) \rceil) \text{ [bits]} \qquad (13)$$

where N denotes the image width, H the height of the SE, bpp the number of bits per pixel, and $\lceil . \rceil$ the ceiling operation. For example, let $N = 1024$ px, $H = 31$ px, and $bpp = 8$ bits. The memory requirement is then

$$R = 31744 \times 13 \text{ [bits]} \qquad (14)$$

The memory occupation of the pipeline is then linear factor of the memory occupation of one stage and the length of the pipeline.

The ALU performs simple dyadic, arithmetic operations on two inputs. Each input can be connected to either an image stream or a programmable constant. The supported operations are as follows: no operation; negation (logical complement); bit-wise AND, OR, XOR; saturated addition, subtraction; infimum and supremum.

In order to ensure that the ALU has both input pixels at the same coordinates in respective images, both image streams have to be synchronized. The synchronization is done at two levels.

1. *Algorithm level*: Consider, e.g. the inner or outer morphological gradient Eqs. 2,3, with B a 3×3 window, or the opening/closing residues Eqs. 6. Computing the dilation introduces a delay in $\delta_B(f)$ of one row plus one column (distance from the center of B to the lower-right

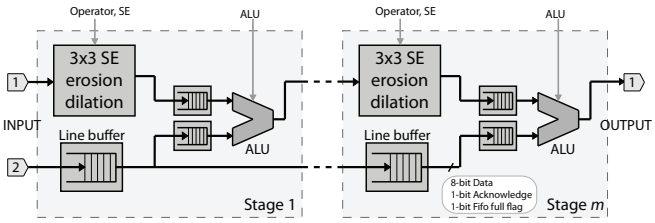


Fig. 2 Geodesic pipeline architecture.

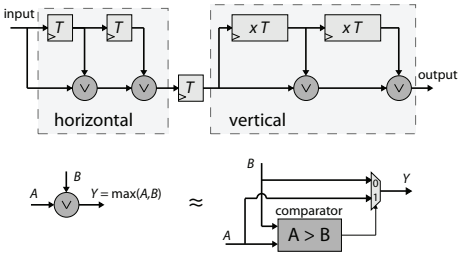


Fig. 3 3×3 dilation unit for the geodesic pipeline.

pixel). This stream f must be delayed accordingly so that the ALU can compute the difference. This is done by turning off the dilation/erosion in the other Large SE block. The computation memory changes into a large FIFO buffer used to delay the stream.

2. *Implementation level*: The previous delay increases with the size of the SE B. Moreover, the dilation/erosion algorithms are asynchronous, see [9] for details. That means that that the delay of the Large SE block can (up to a few pixels) vary. This small delay is compensated by the FIFO memories. In the case that either FIFO is temporarily empty, the empy FIFO stalls the ALU until data become available again.

This synchronization mechanism is automatic, and remains hidden to the user.

The Measurement unit computes simple metrics of the whole image, namely the sum, infimum, and supremum. This measurement is useful in image analysis applications, such as pattern spectrum, and can be obtained on-the-fly at a low cost. The measurement results can be read out through the configuration registers.

All programmable parameters including the SE dimensions, operation, multiplexers and ALU settings, as well as measurement results, are stored in a bank of per-stage configuration registers.

4.2 Geodesic pipeline

A significant subset of morphological operators relies on the reconstruction using the geodesic dilation/erosion by 3×3 SE. Even though the Large SE pipeline supports geodesic operations, using it would be inefficient. A better solution is to devise a dedicated Geodesic pipeline, see Fig. 2.

This Geodesic pipeline contains several equal stages connected one after another. The pipeline is scalable by means of the number of stages instantiated, which is hereafter denoted as m. The heart of each stage is a 3×3 erosion/dilation unit. The output of this unit is connected to the ALU, along with the buffered Mask image. The ALU result is the Marker input of the next stage.

We have used a m=16-stage pipeline in our application. The length of the pipeline is a trade-off between the memory occupation, and the number of iterations before idempotence of a geodesic operation. Given that, geodesic operations require content-dependent number of iterations it is beneficial to have longer pipelines to reduce the number of iterations through the memory.

The 3×3 dilation is outlined in Fig. 3. It also takes advantage of separability of the rectangle into the horizontal and vertical segments, which are implemented using a well-known approach of delay elements (registers T for the horizontal segment, and line buffers xT for the vertical segments) and comparators. The T registers provide a unitary delay, whereas the xT line buffers provide one-image-line delay each. The xT line buffers are parametrized by the the image width to adapt automatically to the image size.

Notice that the delay-line dilation is better adapted for small (3×3 SEs) because it has a smaller area occupation and it is synchronnous. Conversely, the queue-based dilation architecture offers better flexibility due to the programmable SE size and is therefore used in Large SE blocks.

The ALU is the same as described above. The reason for the Line buffer is to synchronize the Mask image and the dilated Marker image, which is delayed by N+1 pixels (recall N is the width of the image).

4.3 Pipeline configurations

The interconnection pattern of the pipelines is thought for maximum polyvalence. The architecture Fig. 4(a), with one, twin Large SE pipeline, and one geodesic pipeline, can be programmed into one of the following patterns (b-f) using global and pipeline multiplexers.

The configuration Fig. 4(b) can be used to compute elementary operators such as the gradient (Eqs. 2-4), an opening, closing (Eqs. 5) or its residues (Eqs. 6). The geodesic pipeline, unused here, appears in grey. The example in Fig. 4(c) connects all the Large SE units into one, long pipeline, suitable for long concatenations of erosions and dilations as in sequential filters (Eqs. 7, 8) or granulometries (Eq. 12). A morphological reconstruction (Eq. 10) is implemented by the geodesic pipeline Fig. 4(d). With both pipelines active and concatenated, as in Fig. 4(e), one can compute openings/closings by reconstruction Eq. 11. The same configuration can also be used in granulometry when the opening by reconstruction Eq. 11 is used Eq. 12.

Such variety of operators allows to fully appreciate the potential of this platform offering a long and flexible pipeline to process data with limited accesses to the memory. Finally, the two pipelines can also be used independently and in parallel on the same input data as in Fig. 4(f).

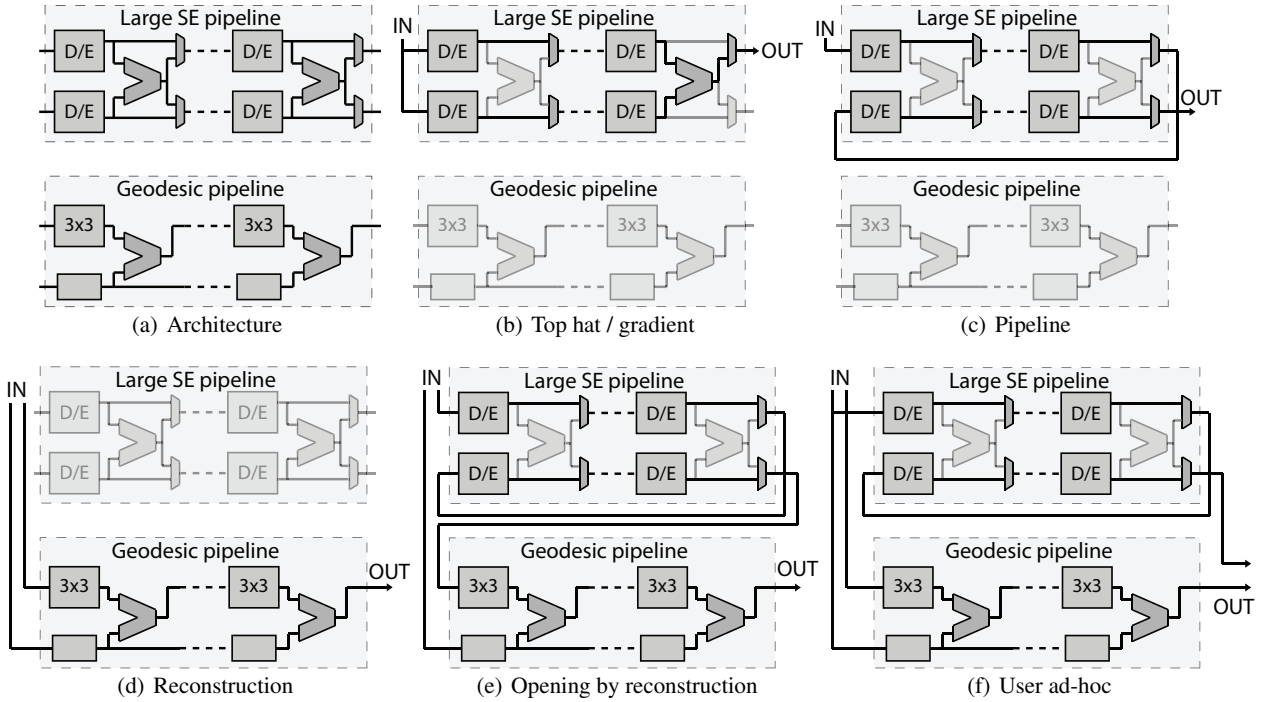


Fig. 4 (a) The interconenction architecture of the pipelines, and (b-f) various possible interconnection patterns of the Large SE and Geodesic pipelines.

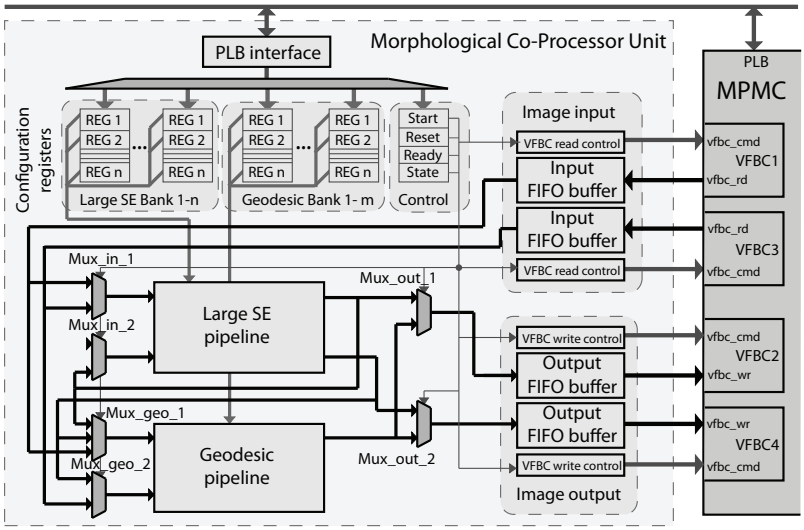


Fig. 5 Architecture of the Morphological Co-Processor Unit. (Legend: Black line denotes an image data bus; grey line denotes configuration and control; MPMC=Multi-Port Memory Controller; VFBC=Video Frame Buffer Controller; PLB=Peripheral Local Bus)

4.4 Morphological Co-Processor Unit (MCPU) Schematics

The pipelines are integrated into the MCPU, supposed to ensure their correct operation, using a set of multiplexers, configuration registers, and image buffers and a memory controller, see Fig. 5.

The configuration registers store the necessary configuration for all the processing units in both pipelines, the global control, and measurement results. Notice that there is one bank of registers for each stage of the processing pipelines. All the registers are accessible to any PLB master by simple read and write instructions.

Image data are transferred by four Video Frame Buffer Controller (VFBC) channels [38]. Two VFBC channels are dedicated to reading input image data from the DDR2 memory, and other two for writing the output image data to the DDR2 memory. The image transfers are independent of each other, so the processing can run in-place (the output image

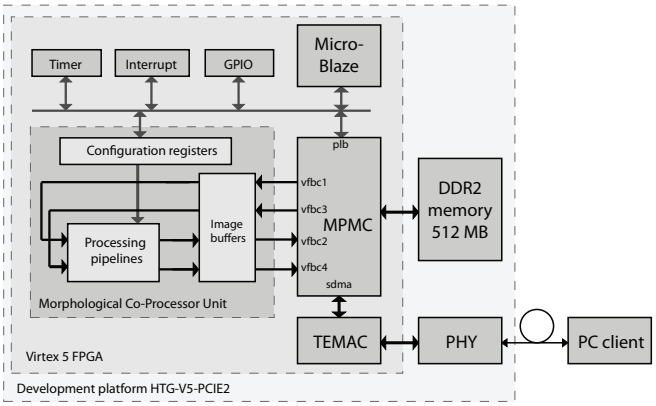


Fig. 6 Overall Platform architecture. Black line denotes image data transfers, grey line denotes configuration and control.

is written in place of the input image). The VFBC allows us to read and/or write image data to the DDR memory with a FIFO-like data-flow control (full, almost full, empty, almost empty flags), so the data stream can be stalled by either endpoint if necessary. The image data in both directions are buffered in Input or Output buffers, respectively.

The MCPU is intended to be used as a peripheral in a higher-level environment. We have tested it as a peripheral of the Xilinx MicroBlaze as described in the following section.

4.5 Top-level architecture

The architecture we have built for evaluation purposes, is outlined in Fig. 6. The proposed MCPU is a coprocessor running as a peripheral of the MicroBlaze CPU synthesized on a Virtex 5 development platform. This platform is also provided with an ethernet link. The architecture consists of two main parts: (i) the MCPU core, and (ii) the MicroBlaze processor environment.

This platform plays several roles: i) the MicroBlaze configures and sends operators to MCPU to execute, ii) provides working memory storage capacity, and iii) handles the communication with the outside world. The images are received via the enternet link and stored in memory.

A very important aspect of every image processing platform is the memory for storing images; either input, output, or intermediate result. The MCPU uses the Xilinx proprietary Multi-Port Memory Controller MPMC [40] that provides a multi-port interface to a high-capacity off-chip DDR2 memory. The MPMC is capable of handling 4 simultaneous image data streams of approx. 50 Mpx/s each that are required by the processing core to sustain the maximal performance.

The MicroBlaze processor uses the Peripheral Local Bus (PLB) to control all the peripherals and to transfer the configuration data, which are small in size, among the peripherals. We used the MicroBlaze version 7.30.b running at 125 MHz with 2048 Bytes of instruction cache and 2048 Bytes of data cache.

5 User interface - programmability

The platform can be accessed via a tri-speed Ethernet interface using either TCP/IP or UDP/IP protocols (implemented as lightweight lwIp). The MCPU runs a server able to accept images and operations to execute via the ethernet link from a superior client.

In order to achieve a proper function of the MCPU server, the client has to perform the following tasks: communicate properly with MicroBlaze, send the image data, configure and run the processing core, read the results.

We have provided a software interface (integrated in MorphM [22]) to handle these tasks. This interface is available at two levels: i) a low-level interface in C/C++, and ii) a high-level interface in C/C++ and Python.

The low-level interface gives the user the possibility to directly control all the features of the computing blocks, such as the sizes of the 1-D segments composing the SE, handle delays, synchronization, multiplexors, turn on/off individual computing units, allocate images in the memory, start and monitor a computation, wait for the end, etc.

The high-level interface offers the user a more intuitive way to execute the most frequent morphological operators. It contains a set of macro functions that hide the low-level programming burden and provide the user the possibility to use the platform on a higher level of abstraction. For example, it can automatically map an arbitrarily long ASF or granulometry to the pipelines. This involves programminc correct SE sizes at every stage, chain the Large SE units in a stream, store the intermediate result in the memory whereever needed, and control the execution and monitor its end.

The MCPU is intended to be as polyvalent as possible, able to perform various operators with different computation schemes. The list of supported operators includes dilation, erosion, opening, closing, reconstruction, opening and closing by reconstruction, top hat, gradient, ASF^{λ}, pattern spectrum, and pattern spectrum by reconstruction. All these operators proceed in the following steps: send the image (if necessary), calculate the configuration based on the passed arguments (either SE or λ), execute the processing pipelines, and read the output image and/or measurement results. Notice that operators with large values of SE or λ may not fit into the pipelines. In this case, several iteration are automatically executed. This is especially true for the reconstruction that may need hundreds of iterations.

Only a few features of the MCPU are set prior to the synthesis, such as the width of the buses or the number of stages in the pipelines. These features remain immuable afterwards. All other parameters are programmable via the configuration registers.

6 Application Example

In this section, we illustrate how MCPU can be used in a real image processing application.

Consider detecting defects in manufactered surfaces, where the defects appear as changes in local textural patterns. Cord *et al.* [7] follow a probabilistic approach, considering textural variations as realizations of random functions. Taking into account information of pixel neighbourhoods, the texture for each pixel is described at different scales. By means of statistical learning, the most relevant textural descriptors are selected for each application.

The preparatory steps of this technique consist of the following (refer to [7] for details and references):

1. A collection of morphological descriptors is chosen, such as dilations, erosions, openings and closings, each for various SE shapes, such as segments oriented in 0°, 45°, 60°, 90°, 120° and 135°, squares and circles and each for different sizes. Each pixel is assigned a vector with as elements the values from these descriptors.
2. A dimension reduction in a set of independent variables by Principal Component Analysis.
3. A supervised learning using Linear Discriminant Analysis.
4. A variable selection based on forward selection.

Done in this way, the approach is generic, and the selected descriptors application specific. After validation, we know at this stage which descriptors apply best to separate the classes – defect/no defect.

The two-step on line classification application shall execute on the MCPU.

1. The vector of values for all selected sizes of a descriptor is a local pattern spectrum with this SE. The descriptors for every SE shape and size are efficiently computed MCPU and the descriptors stored in the DDR2 memory.
2. The LDA classification is a weighted linear combination of the descriptor values for every pixel which, coded in C, can run on the MicroBlaze controller. The resulting image is then trasferred out via the ethernet link.

7 Experiment setup

The proposed MCPU architecture has been implemented in VHDL and targeted to the Xilinx platform HTG-V5-PCIE2, equipped with one medium-size Xilinx Virtex-5 FPGA chip XC5VSX95T [39]. We report here two different setups with parameters to fully utilize the available FPGA resources and keep high flexibility at the same time. Using the XC5VSX95T – a medium-size Virtex 5 family FPGA, with 14,720 slices and 244 36Kb-size RAM blocks – we could implement 4 to 5 (resp.) Large SE stages and 16 geodesic stages in two pipelines on the chip. The implementation results and setup specification are outlined in Table 1. Notice that the reported "Supported SE per unit" is given by the maximum FIFO length as synthesized on the chip. After the synthesis, the user can freely program his own SE size withing this range.

Table 1 Implementation results, Xilinx Virtex-5 FPGA XC5VSX95T [39]

Parameter	Setup 1	Setup 2
Supported SE per unit	Rectangle 31×31	Octagon 43×43
Large SE stages n	5	4
Geodesic stages m	16	16
Image width N	1024	1000
FPGA Slice	13534	14684
FPGA BRAM	233	243
Clock frequency	125 MHz	125 MHz

Recall that the SE size multiplied by the image width is the determinant factor of the memory occupation per Large SE stage, ref Eq. 13. The memory occupation for the Large SE pipeline is linear factor of its length. The architecture is therefore linearly scalable – factor of the image width, SE size and number of stages (and also bits per pixel) withing the limitation of the FPGA size.

8 Performance Comparison

In this section we compare this architecture with other embedded solutions: software solutions (Sec. 8.1) and existing hardware platforms (Sec. 8.2).

8.1 Software solutions

We compare the performance of the proposed architecture against another embedded solution, an ARM processor. In our case, we use the Sabre platform [11] by Freescale, which can be seen as another example of hand-held platform. The Sabre uses quad-core ARM A9 processor at 1GHz, 1GB of DDR3 memory up to 533MHz, and runs Linux with TCP/IP stack. We have created benchmarks for two image processing libraries: (i) the well-known OpenCV [24], and (ii) highly optimized Smil [20]. For the sake of completeness, we have also included the single-thread results of the OpenCV at desktop PC Intel Xeon E5620, 2.40 GHz, with 24 GB memory, running a Fedora, release 20, linux.

The benchmark in Table 2 includes a set of aforementioned morphological operators on natural gray-scale photos 1000×1000 px. It includes an elementary 3×3 dilation, large opening and opening by reconstruction, alternating sequential filter, pattern spectrum and pattern spectrum by reconstruction. Apart from the 3×3 dilation, the common property of all these operators is the large number of operations, which is even undetermined for the reconstruction, and therefore, a high cost.

The experimental results show that the proposed MCPU architecture delivers performance by orders of magnitude superior to that of the Sabre platform, and even comparable

Table 2 Performance results of selected operators. Image is natural photo 1000×1000 px, time results are in milliseconds (unless seconds are specified).

Operator	Shape of SE	Size of SE or λ	MCPU	OpenCV at Sabre	Smil at Sabre	OpenCV at Xeon
Dilation	Rectangle	3×3	21.9	32.7	8.4	0.58
Opening	Rectangle	151×151	24.3	2450	1083	38.6
Opening	Octagon	151×151	41.9	246 s	2453	2301
Opening by recon.	Rectangle	151×151	544	47.6 s	22.1 s / 2110*	1940
Opening by recon.	Octagon	151×151	512	289 s	21.1 s	4356
ASF	Rectangle	$\lambda = 11$	64.2	4530	1987	57.1
ASF	Octagon	$\lambda = 11$	83.3	77 s	3872	814
Pattern spectrum *PS*	Rectangle	$\lambda = 11$	62.3	2570	1098	53.8
Pattern spectrum *PS*	Octagon	$\lambda = 11$	62.7	21.2 s	1782	249
PS by recon.	Rectangle	$\lambda = 11$	2530	190 s	85.3 s / 18.2 s*	8920
PS by recon.	Octagon	$\lambda = 11$	2410	201 s	81.5 s	8751

note *: The second result is obtained by an algorithm based on hierarchy queues.

Table 3 Comparison of several FPGA and ASIC architectures concerning morphological dilation and erosion. N, M stand for the image width and height of respective architectures.

	Processing unit				Hardware System		Application Example ASF^6		
	Technology	Supported SE	Throughput [Mpx/s]	f_{max} [MHz]	Number of units	Supported image	Image scans	FPS [frame/s]	Latency [px]
Clienti [4]	FPGA	arbitrary 3×3	403	100	16	1024×1024	6	66.7	$5NM + 84N$
Chien [3]	ASIC	disc 5×5	190	200	1	720×480	45	12.2	$44NM + 84N$
Déforges (a) [8]	FPGA	arbitrary 8-convex	50	50	1	512×512	13	14.7	$12NM + 84N$
Déforges (b) [8]	FPGA	arbitrary 8-convex	50	50	13	512×512	1	50	$84N$
This paper	FPGA	regular polygon	195	100	13	1024×1024	1	185	$84N$

with a desktop PC, for all high-cost operations, i.e., all in Table 2 but the 3×3 dilation. MCPU outperforms the other platforms (or is at least equivalent) whereever a high number of operators are sequentially applied to the image. Such a significant speed-up is allowed by possibility to thoroughly exploit the inter-operator parallelism via the pipelined computation. This is especially true for the opening and pattern spectrum by reconstruction when $m = 16$ geodesic dilations are computed at the same time. The speed-up becomes less significant for simple operators with small SEs, the performance for 3×3 dilation is worse than that of Smil at Sabre. This is due to a much higher clock of the ARM and the Xeon processors (1GHz and 2.40 GHz, respectively). However, the majority of applications of mathematical morphology need a long sequence of operators that take advantage of the proposed parallelism.

Evaluated using the Xilinx Power Estimator tool [37], the total on-chip power consumption is 4.424 W (1.237 W for input/output ports, 1.625 W of dynamic and 1.562 W of static consumption). The consumption of the Sabre platform is ≈3 W during intensive workload [10]. The thermal design power of the Intel Xeon E5620, 2.40 GHz CPU is ≈80 W during intesive workload [16].

8.2 Existing HW solutions

The Table 3 offers a comparison of our architecture with a few others that support flat, non-rectangular SE. The Processing Unit section of this table provides a comparison on the basis on a single 2-D δ/ε unit only. Clienti [4] yields a high throughput for an elementary SE 3×3. The Chien [3] ASIC chip achieves a reasonable throughput with a small 5×5 diamond SE. The Déforges [8] unit supports 8-convex SEs decomposed as a concatenation of elementary 2-pixel SE. (Lower throughput is probably due to a less powerfull device than that of the other implementations.)

For all other architectures (except this paper) the flexibility to control the size and shape of the SE after the synthesis remains unclear. Nonetheless, all architectures can use the homothecy to obtain larger SEs. This requires using a long processing pipeline as in Clienti [4] or Déforge *et al.* [8](b). Iterating over the memory significantly decreases the overall throughput.

The Hardware System section of the table lists the number of units synthesized on the chip. The performance of every architecture for an ASF^6 as example is given in the third section of the table. The FPS decreases drastically as the number of iterations over the image increases, as for Chien [3] or a single unit of Déforges *et al.* [8]. The decrease is less significant for Clienti [4] who uses a 16-unit long pipeline. However, the units are small 3×3, and still 6 images scans are needed.

Only the Déforges *et al.* [8] pipeline and this paper architecture can embed the entire ASF with no or a limited decrease in performance.

9 Conclusions

This paper proposes a novel programmable morphological coprocessor for embedded devices based on FPGA devices. We have integrated previously published efficient dilation/erosion processing units and geodesic units into a MicroBlaze platform, which provides DDR memory storage and Ethernet connectivity, and thus created a very powerful coprocessor that supports a wide range of operators from a simple dilation to the pattern spectrum by reconstruction.

The coprocessor was experimentally evaluated at a Virtex5 development kit and compared to the quad-core ARM9 Sabre platform by Freescale running OpenCV and Smil libraries. The performance results for various compound operators (except the 3×3 dilation) show a significant speed-up of at least one order of magnitude. The results of MCPU do even compare to that of a Xeon desktop workstation.

The future work will be focused on development of a compiler for MCPU that will automatically map a given application to the architecture. The current interface provides a user with a high-level programming interface. In the future, this compiler shall optimize the execution of concurrent operators, branching and simultaneous co-execution of an application on MCPU and the client.

References

1. J. Bartovský, P. Dokládal, E. Dokládalová, M. Bilodeau, and M. Akil. Real-time implementation of morphological filters with polygonal structuring elements. *Journal of Real-Time Image Processing*, 10(1):175–187, 2012.
2. J. Bartovský, P. Dokládal, E. Dokládalová, and V. Georgiev. Parallel implementation of sequential morphological filters. *Journal of Real-Time Image Processing*, 9(2):315–327, 2014.
3. S.-Y. Chien, S.-Y. Ma, and L.-G. Chen. Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1156 – 1169, sept. 2005.
4. Ch. Clienti, S. Beucher, and M. Bilodeau. A system on chip dedicated to pipeline neighborhood processing for mathematical morphology. In EURASIP, editor, *EUSIPCO 2008*, Lausanne, August 2008.
5. Ch. Clienti, M. Bilodeau, and S. Beucher. An efficient hardware architecture without line memories for morphological image processing. In *ACIVS '08*, pages 147–156, Berlin, Heidelberg, 2008. Springer-Verlag.
6. D. Coltuc and I. Pitas. On fast running max-min filtering. *IEEE Transactions on Circuits and Systems II*, 44(8):660 –663, aug 1997.
7. A. Cord, F. Bach, and D. Jeulin. Texture classification by statistical learning from morphological image processing: application to metallic surfaces. *J. of Microscopy*, 239:159–166, 2010.
8. O. Déforges, N. Normand, and M. Babel. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. *Journal of Real-Time Image Processing*, pages 1–10, 2010.
9. P. Dokládal and E. Dokládalová. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22(5):411–420, 2011.
10. Freescale. i.MX 6Dual/6Quad Power Consumption Measurement, Rev. 0, 10/2012. http://cache.freescale.com/files/32bit/doc/app_note/AN4509.pdf.
11. Freescale. SABRE reference designs, 2014. http://www.freescale.com/sabre.
12. R. M. Gibson, A. Ahmadinia, S. G. McMeekin, N. C. Strang, and G. Morison. A reconfigurable real-time morphological system for augmented vision. *EURASIP Journal on Advances in Signal Processing*, 2013(1), 2013.
13. J. Gil and R. Kimmel. Efficient dilation, erosion, opening, and closing algorithms. *IEEE Trans. PAMI*, 24(12):1606–1617, 2002.
14. J. Gil and M. Werman. Computing 2-d min, median, and max filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):504–507, 1993.
15. M. Holzer, F. Schumacher, T. Greiner, and W. Rosenstiel. Optimized hardware architecture of a smart camera with novel cyclic image line storage structures for morphological raster scan image processing. In *Emerging Signal Processing Applications (ESPA), 2012 IEEE International Conference on*, pages 83–86, Jan 2012.
16. Intel. Intel Xeon Processor E5620 (12M Cache, 2.40 GHz, 5.86 GT/s Intel QPI). http://ark.intel.com/products/47925/Intel-Xeon-Processor-E5620-12M-Cache-2_40-GHz-5_86-GTs-Intel-QPI.
17. J.-C. Klein and J. Serra. The texture analyser. *J. of Microscopy*, 95:349–356, 1972.
18. D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *CoRR*, abs/cs/0610046, 2006.
19. F. Lemonnier and J.-C. Klein. Fast dilation by large 1D structuring elements. In *Proc. Int. Workshop Nonlinear Signal and Img. Proc.*, pages 479–482, Greece, Jun. 1995.
20. Faessel M. Smil simple morphological image library. http://smil.cmm.mines-paristech.fr, 2014.
21. G. Matheron. *Random sets and integral geometry*. Wiley New York, 1975.
22. Morph-M. Morph-M documentation. http://cmm.ensmp.fr/Morph-M, 2012.
23. N. Normand. Convex structuring element decomposition for single scan binary mathematical morphology. In *Discrete Geometry for Computer Imagery*, volume 2886 of *LNCS*, pages 154–163. Springer Berlin, Heidelberg, 2003.
24. OpenCV. OpenCV documentation. http://opencv.org, 2014.
25. J Pecht. Speeding-up successive minkowski operations with bit-plane computers. *Pattern Recognition Letters*, 3(2):113 – 117, 1985.
26. I. Pitas. Fast algorithms for running ordering and max/min calculation. *Circuits and Systems, IEEE Transactions on*, 36(6):795 –804, June 1989.
27. J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.
28. J. Serra. *Image Analysis and Mathematical Morphology, Volume 2, Theoretical Advances*. Academic Press, London, 1988.
29. J. Serra and L. Vincent. An overview of morphological filtering. *Circuits Syst. Signal Process.*, 11(1):47–108, 1992.
30. P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
31. P. Soille, E. J. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(5):562–567, 1996.
32. C. Torres-Huitzil. FPGA-based fast computation of gray-level morphological granulometries. *Journal of Real-Time Image Processing*, pages 1–11, 2013.
33. E. R. Urbach and M. H. F. Wilkinson. Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements. *IEEE Trans. Image Processing*, 17(1):1 –8, jan. 2008.
34. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recogn. Lett.*, 13(7):517–521, 1992.
35. J. Velten and A. Kummert. Implementation of a high-performance hardware architecture for binary morphological image processing operations. In *Circuits and Systems, 2004. MWSCAS '04. The 2004 47th Midwest Symposium on*, volume 2, pages II–241 – II–244 vol.2, 25-28 2004.
36. L. Vincent. Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. *Image Processing, IEEE Transactions on*, 2(2):176–201, Apr 1993.

37. Xilinx. Xilinx Power Estimator (XPE). http://www.xilinx.com/products/design_tools/logic_design/xpe.htm.
38. Xilinx. Video Frame Buffer Controller v1.0. http://www.xilinx.com/products/devboards/reference_design/vsk_s3/vfbc_xmp013.pdf, October 29 2007.
39. Xilinx. Virtex-5 family overview. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, February 6 2009.
40. Xilinx. LogiCORE IP Multi-Port Memory Controller (MPMC) (v6.03.a). http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf, March 1 2011.
41. J. Xu. Decomposition of convex polygonal morphological structuring elements into neighborhood subsets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(2):153–162, 1991.

EURASIP Journal on Applied Signal Processing 2005:17, 2788–2803

Embedded Real-Time Architecture for Level-Set-Based Active Contours

Eva Dejnožková

Centre of Mathematical Morphology, School of Mines of Paris, 35 Rue Saint Honoré, 77305 Fontainebleau Cedex, France
Email: dejnozke@cmm.ensmp.fr

Petr Dokládal

Centre of Mathematical Morphology, School of Mines of Paris, 35 Rue Saint Honoré, 77305 Fontainebleau Cedex, France
Email: dokladal@cmm.ensmp.fr

Received 14 June 2004; Revised 5 April 2005; Recommended for Publication by Luciano da F. Costa

Methods described by partial differential equations have gained a considerable interest because of undoubtful advantages such as an easy mathematical description of the underlying physics phenomena, subpixel precision, isotropy, or direct extension to higher dimensions. Though their implementation within the level set framework offers other interesting advantages, their vast industrial deployment on embedded systems is slowed down by their considerable computational effort. This paper exploits the high parallelization potential of the operators from the level set framework and proposes a scalable, asynchronous, multiprocessor platform suitable for system-on-chip solutions. We concentrate on obtaining real-time execution capabilities. The performance is evaluated on a continuous watershed and an object-tracking application based on a simple gradient-based attraction force driving the active countour. The proposed architecture can be realized on commercially available FPGAs. It is built around general-purpose processor cores, and can run code developed with usual tools.

Keywords and phrases: level set, partial differential equations, object tracking, real-time execution, embedded platforms.

1. INTRODUCTION

The level set was proposed in 1988 in [1] as a simple method to modelize or analyze the motion of a travelling interface. It offers a convenient and stable framework to implement a large variety of methods where images are seen as sets of curves. Since then, its applications have been extended to other image processing fields such as the restoration (filtering or contrast enhancement), segmentation (active contours, watershed) to the form analysis (shortest path, shape-from-shading). See [2] or textbooks [3, 4] for applications and a general overview.

From the implementational point of view, the methods can be divided into two groups: (i) filtering-like methods operating on a set of constant-level curves describing the entire image and (ii) methods that act on a single (or several) contour(s), representing one (or several) object(s) present in the image. Below, we reference these algorithms according to their computation scope, the filtering-like methods as *global-scope*-type and the active contours methods as *narrowband* type.

1.1. Scope and objectives

The objective of this paper is to open the world of hand-held, mobile devices such as PDAs, still picture or movie cameras or mobile phones to powerful image processing methods from the level set framework. The novelty of this paper resides in the presentation of a reusable architecture capable to run optimally various algorithm types from the level set family. This architecture corresponds well to the system-on-chip concept, and verifies the needs of hand-held devices concerning their energy and implementational limitations. We particularly concentrate, among other aspects, on the execution on multiprocessor, parallel, scalable architectures which is an important aspect permitting to reduce the energic consumption.

The rest of this paper is organized as follow. After reviewing the state of the art of existing implementations and accelleration attempts, analyzing the family of the level-set-based algorithms (Section 2), we present the architecture (in Section 3) that best verifies the algorithmic needs and remains efficient with respect to the HW implementation issues listed above. Its efficiency is demonstrated on a contour-tracking algorithm proposed in Section 4.2. The text concludes by presenting some benchmark results and general conclusions.

1.2. State of the art and technological difficulties

The implicit representation of the travelling interface by using the level set increases the computational effort by one

order of magnitude. A faster implementation obtained by narrowbanding the computations around the travelling interface (originally called the *tube method*) was proposed by Adalsteinsson *et al.* [5] and Malladi *et al.* [6]. Despite the narrowbanding which reduces considerably the number of points to process, the level set methods remain computationally expensive, because of (i) using nonlinear functions, and (ii) a high number of iterations. The computational complexity has unpleasant consequences on both the execution time and the power consumption. If the execution time can be reduced by parallel execution (provided that the algorithm is parallelizable), the overall energy budget (following from the number of necessary operations multiplied by the energy to perform one basic operation) remains constant.

The numerous attempts to speed up the implementation of PDE-based methods made in the past were done in various axes.

(i) Algorithmic: as, for example, the implementation alternative to the level set, using spline-based modeling of the contours. Precioso and Barlaud [7] have obtained a fast execution on Pentium-based machines with spline-based active contours. A special care must be done to handle the topology changes. Cserey et al. study in [8] the implementation of linear and nonlinear diffusions on neural networks. In general, the algorithmic modifications are often applicable to only one type of algorithms.

(ii) Mathematical: proposing a faster convergence either in another space, or using another integration scheme. Weickert et al. propose in [9] the semi-implicit integration scheme, and the AOS scheme with arbitrarily large integration step for filters which can be written in a specific form as in [10]. The semi-implicit scheme increases the integration speed without affecting the numerical stability; it deteriorates only the numerical accuracy. Later, Goldenberg *et al.* [11] and Smereka [12] use a semi-implicit scheme for the active contours. However, the mathematical modifications are often applicable to only a restricted family of algorithms.

(iii) Hardware-based implementations are of three types.

(a) *Supercomputers*: Holmgren and Wallin [13] use a self-optimizing nonuniform memory access (NUMA) supercomputer implementing a high-accuracy solver for several integration kernels. Sethian [14] has studied study flame propagation models on a CM-2 machine with 65K processors. The author reports a true massively parallel calculation with one processor per grid node.

(b) *Graphic hardware*: Rumpf and Strzodka benefit from a high memory bandwidth and implement a nonlinear diffusion [15], and a level set segmentation [16] on a graphic card. Cates *et al.* [17] implement an active-contours-based segmentation tool on a graphic hardware to increase the interactivity when a number of parameters must be tuned to obtain a correct segmentation results. Sigg *et al.* implement a signed-distance function transform on a graphic hardware [18].

(c) *Specific HW accelerators*: Hwang *et al.* [19] propose an orthogonal architecture designed for numerical solution of PDEs, not inevitably related to the image processing. It is built around n processing units and n^2 memory blocks. Each processor is connected to the memories by buses dedicated to only one processor, equipped with a memory access controller. The drawback of this design is that the number of interconnexions and buses increases with the square of the number of processors.

Gijbels *et al.* [20] propose a VLSI architecture for nonlinear diffusion conceived for image improvement on image sequences. The authors use an SIMD[1] architecture with distributed memory for parallel nonlinear diffusion (i.e., the global-scope-type) used in some vision application. The estimated performances are some 100 iterations on a 256×256 image every 0.25 seconds, whereas the processing units themselves are clocked at 20 MHz.

This paper focuses on the HW-based implementation issues of the level set techniques on embedded, *one-chip devices* that will be easily (i) *scalable*, to adapt their computational power to the requirements of the chosen application, (ii) *programmable* with conventional programming tools, (iii) by far *less energy consuming* than Pentium-based desktop machines with comparable computational power, and (iv) as *small sized* as possible. The surface occupation is important because it has a direct impact on the price of both the chip itself and the embedding system (such as personal vehicles, hand-held devices, etc.). These contraints exclude both the graphic hardware and supercomputer implementations, since they do not match the objectives of one-chip devices, as well as the SIMD architecture, presented in [20], which cannot be used either because of its considerable number of used processing units (one unit per image column).

Generally speaking, it is essentially due to the algorithmic complexity that no embedded platforms have so far been proposed for the narrowband-type algorithms. The issues to handle include the following.

(i) *Nonlinear computations* employed in the integration step, *numerous iterations* necessary to obtain the convergence, and often required *floating-point accuracy* impose using fast ALUs. Their considerable surface occupation and energy consumption exclude their replication in a great number on one chip.

(ii) The *distance function* computation represents another difficulty of parallelization of the narrowband applications. Dejnožková and Dokládal [21] present a detailed analysis of existing algorithms (namely fast march-

[1] Single instruction multiple data.

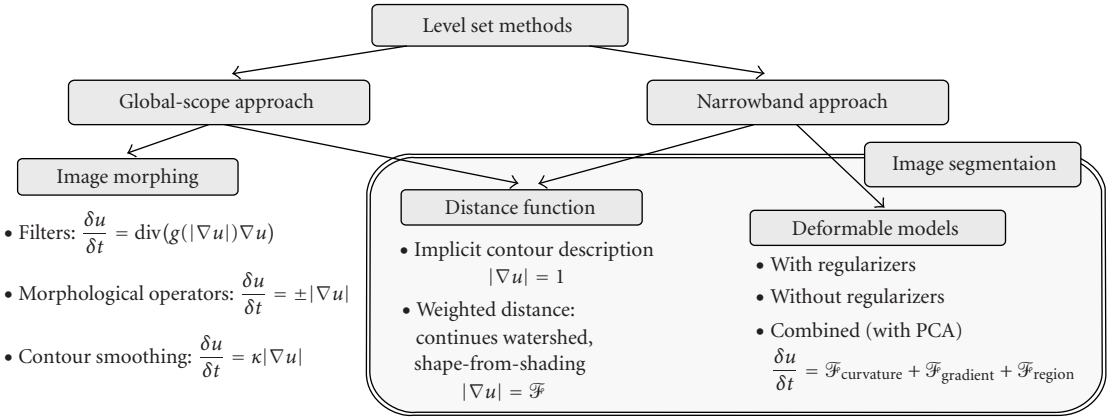


Figure 1: PDE-based algorithms overview.

ing). It shows that they are sequential and ordered (explained below). The authors propose to remove the bottleneck with the introduction of massive marching [21]. It is a fully parallel algorithm, making use of a nonequidistant propagation front.

Since we aim the entire algorithm family, whose common denominator is the level set implementation, the architecture has to be maximally flexible and scalable, and maximally using the occupied silicon surface. The recently emerging dynamic reconfiguration represents an alternative solution to the tradeoff between the functional flexibility of complex systems and the occupied surface and energy consumption, see for example [22, 23]. In some cases, the advantages of the dynamic reconfiguration may however be outmatched by the drawbacks that constitute a lenghty and difficult design, the need for special design tools [24], and external circuits controlling the chip reconfiguration.

Our study demonstrates that for the level set domain, the satisfying tradeoff between the flexibility and size can also be obtained by the programmability, offered by on-chip embedded processor cores and some DSP functions.

The following section presents the analysis of the architectural choices, including the computational resources, memory consistency model, and communication management. The resulting system has been synthesized for commercially available FPGAs.[2] Their performance becomes almost comparable to the ASICs.[3] Though the ASICs still outperform the FPGAs in the energy consumption (a key feature in mobile devices), the FPGAs remain a useful prototyping platform, and a possible intermediate development step towards an ASIC.

2. ALGORITHM ANALYSIS

This section discusses hardware implementation issues of several algorithm types from the level set context. All the types consist of two basic steps: an *initialization* step that differs according to the method used, and the *evolution* step, which makes the contour(s) travel in space and/or time according to the given partial differential equation (PDE). Usually it makes use of some local integration kernel, and is repeated until stability. In general, only the use of a local information is easily parallelizable. If the image is considered as a continuous signal, then the PDEs can be seen as an iteration of a local filter operating on the neighborhood [4].

Typically, the evolution proceeds by deforming one or several curves (propagation front) or surface with a given PDE. The PDEs methods can be classified into the following categories (cf. Figure 1).

(1) Surface propagation includes *diffusion filters* [25], [26], or [27] for a more comprehensive survey, *geometric smoothing* [10, 28, 29], *denoising*, and *morphological operators* [30], [31], [32] or [33] characterized by the evolution equation $\partial u/\partial t = \mathcal{F}(u)|\nabla u|$, where u represents the evolving image. The input image represents the initial conditions u_0. All points in the image are processed in every iteration. The temporal evolution is based on the local neighborhood and generates the evolution of the level sets in the space [4]. The evolution stops as soon the convergence or the given iteration number is reached.

(2) Wave propagation includes algorithms of *weighted distance*, *continuous watershed* [34], *Voronoi tesselations* [35], or *shape-from-shading* [36] that are controlled by the Eikonal equation $|\nabla u| = \mathcal{F}$. This steady-state solution is propagated from the given sources (that may be obtained from the initial image by other means) on the entire image according to the defined speed $\mathcal{F}$. The algorithm operates locally, only on the narrowband of the evolving front. The solution is propagated in waves equidistant to the sources by using ordered data structures. This technique is being referred to as marching methods, proposed by Sethian [37], as a special case of the Dijkstra shortest-path algorithm.

(3) Deformable models. An important breakthrough in the deformable models represents the introduction of

[2] Field-programmable gate array.
[3] Application-specific integrated circuit.

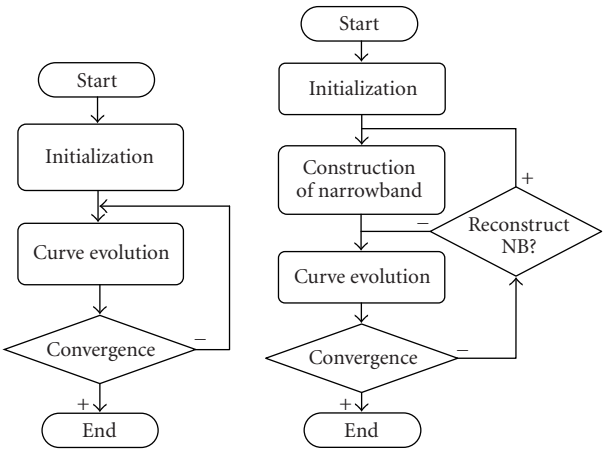


Figure 2: Different stages of the level set family algorithms. (a) *Global-scope*-type algorithms. (b) *Narrowband*-type algorithms.

active contours (or snakes) proposed by Kass *et al.* [38] in 1987, and deformable surfaces by Terzopoulos *et al.* [39] one year later. Another early example of deformable models represents the ballons by Cohen [40]. Implicit representation of the interface as a constant level set of another function was studied simultaneously and independently in 1993 by Caselles *et al.* [41] and Malladi *et al.* [42], and later by Malladi *et al.* [43, 44, 45]. The geodesic active contours were proposed meanwhile in [46, 47]. Another model was proposed later in [48].

We distinguish the types *with regularizers* (controlled by statistical information of regions), *without regularizers* [27], or *combined* with other techniques (e.g., principal component analysis [49]). The evolution equation writes in the form $\partial u/\partial t = \mathcal{F}_{curvature}(u) + \mathcal{F}_{grad}(u) + \mathcal{F}_{region}(u)$. The algorithms proceed by deforming a given initial contour (given by $u_0 = 0$). The deformation is controlled by internal and external forces obtained at each iteration from (i) the contour itself and (ii) the geometrical (curvature, gradient) or statistical characteristics (mean value of the region intensity) found in the image [4].

(4) Optical flow is controlled by the equations $\partial u/\partial t = f(\nabla u, I_1) + g((\partial I_2/\partial x), h)$, $\partial v/\partial t = f(\nabla v, I_1) + g((\partial I_2/\partial y), h)$. The motion vector is obtained by solving some system of the above-given equations at each point in the image (I_1, I_2 are the successive sequence images, h is the searched motion vector field) [50].

Since the nature of the optical flow algorithms differs from the temporal curve evolution principle of the three first groups, the proposed architecture does not address this type of algorithms. On the other hand, the optical flow often serves as a support for the three other types.

All the computation steps of the first three categories can be unified in two following iteration types, see Figure 2. (1) *Global-scope iteration type* includes the surface evolution. It operates *sequentially* on the entire image. (2) *Narrowband iteration type* includes the wave propagation and deformable models (curve evolution).

Indeed, applying narrowbanding to the curve evolution algorithms changes the computational aspects. The points to recalculate in every iteration are now taken from some subset of the image. This set is commonly called narrowband, and contains points situated closely (up to some chosen distance) to the current position of the travelling interface. Two types of operations are commonly applied on the narrowband: (i) the curve motion scheme itself, and (ii) the (re-)construction of the narrowband. The (re-)construction differs substantially from the other algorithm types. Indeed, all HW implementations of the active contours, cited in Section 1, use fast marching; a progressive, equidistant construction of the distance function. Fast marching itself belongs to the *wave propagation* algorithm group. It requires ordered data structures based on the priority of points [3]. From the algorithmical point of view, the ordering introduces a great data dependency, reducing the parallelization potential. From the HW implementation point of view, algorithmic ordering of the points to process introduces *random* accessing to the memory.

Parallelize the wave propagation is a tough issue, calling attention of many researches for a long time, compare a survey by Roerdink and Meijster in [51]. The recent introduction of massive marching opens the possibility to parallelize also the computation of the distance function (cf. [52] or [21]). Massive marching is similar to the fast marching method (cf. [53] or [54]) and uses the same entropy-satisfying upwind scheme. It differs from fast marching by the fact that it eliminates its sorted propagation of the solution and makes the implementation fully parallelizable, with a small grain and low data dependency.

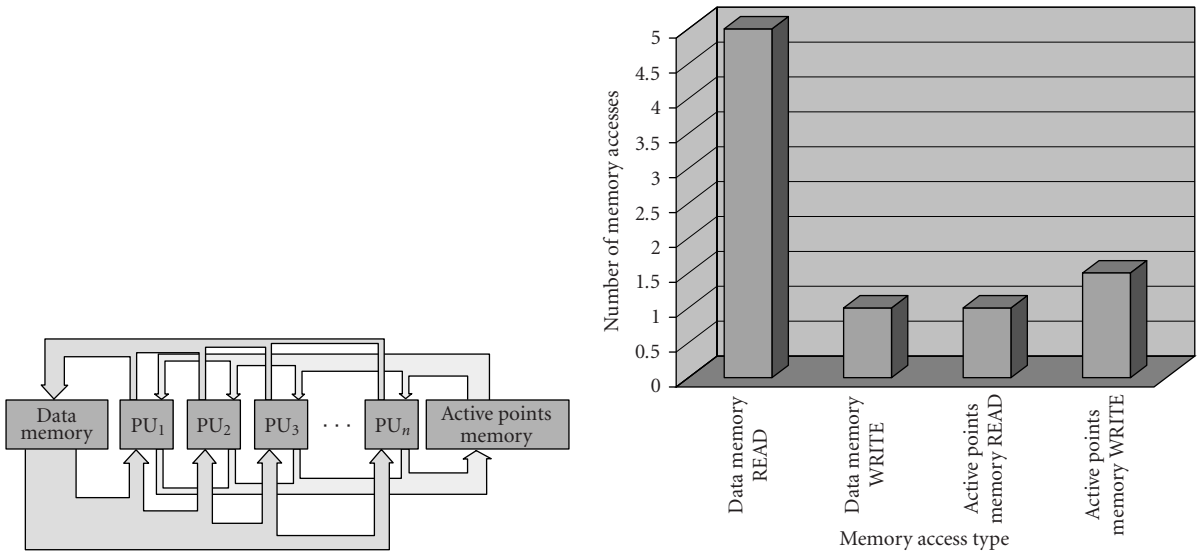
For completeness, we mention another parallelization strategy, called group marching, developed by Kim in [55]. Group marching identifies on the front groups of points that are processed parallely in the same time. It requires nonetheless to maintain a global variable making a truly parallel implementation difficult.

The next section analyzes the execution of the different algorithm steps by considering the use of massive marching for the narrowband construction. Note that the curve evolution (both algorithm types) as well as the narrowband construction (narrowband type) are time-critical. Many iterations may be needed to obtain the convergence and the narrowband has to be reconstructed repetitively during the evolution to preserve the required properties of the implicit curve description.

2.1. Data-flow analysis of different algorithm steps

In the following, we assume that, except the methods where regularizers[4] are used, the new values that the points re-

[4]Statistical information, like colour for example, represents global variables.



(a) (b)

Figure 3: Curve evolution implemented by using several processing units operating in parallel: (a) data-flow chart, and (b) corresponding memory accesses.

ceive are results of operations only on local neighborhood.

Limiting the calculations to a narrowband around the travelling contour corresponds formally to operating on sparse matrices. Practically, in order to obtain the correct evolution, all active points need to be recalculated in one iteration, before the next iteration starts. Therefore, unless one uses one processing unit per point, the values u^{n+1} need to be stored separately from u^n, until all the points are updated. If this condition is verified, then the processing order is not important, and both the iteration types can be unified under the following form. The set $\mathcal{A}$, respectively, represents either the entire image or the narrowband set:

for all $p_i \in \mathcal{A}$ do *(in parallel)*
{ *Retrieve_Neighborhood* $u^n(N(p_i))$ and $u^n(p_i)$;
Calculate_Value $u^{n+1}(p_i)$;
Update_Value $u^{n+1}(p_i)$;
Activate_New_Points (insertion in $\mathcal{A}$); }

Since our constraints exclude the massive parallelism (for production cost's reasons), we adopt a semiparallel approach instead. The data-flow chart corresponding to a semiparallel execution of this code on several processing units is given by Figure 3a. The data u are stored in the data memory block (two pages for u^n and u^{n+1}). The active points memory block stores the set $\mathcal{A}$, that is, the coordinates of the points to process (not used for the global scope-type algorithms). The active points are read and processed by several independently operating processing units. Both the memory blocks are organized in two pages, for the present one and the next iteration.

The width of the paths corresponds to the volumes of transferred data. The most intensive data traffic is on the shared blocks. The READ data memory flow is five times larger than the WRITE data memory flow because the complete four-neighborhood is read to update the central value (cf. Figure 3b). Similarly, since one processed point may activate several of its neighbors, the mean WRITE active points memory flow is slightly higher than READ active points memory.

The narrowbanding of active contours techniques impose random memory access to the data memory block. These aspects will be taken into account in Section 3.

2.2. Timing analysis

To optimize the data flow, limit simultaneous accesses to the shared blocks, and obtain a balanced activity of all the used blocks, it is necessary to consider also the timing of the algorithm execution.

The global-scope type operates on the entire image, that is, each point in the image is active and the set $\mathcal{A} = \text{supp}(I)$, $I =$ image. The narrowband type operates on $\mathcal{A} = \{p \mid |\text{dist}(p)| < NB_{\text{width}}/2\}$, where NB_{width} is the width of the narrowband around the contour. For massive marching, the definition of $\mathcal{A}$ slightly differs (see [21]).

This code has two major features.

(1) The retrieval of the point's and its neighbors' values $u^n(p_i)$ and $u^n(N_4(p_i))$ requires five memory readings and is usually faster than the following calculation of $u^{n+1}(p_i)$, which usually involves nonlinear functions. During the calculation of $u^{n+1}(p_i)$, the memory block is idle.

(2) The execution of same parts of the code can have different length due to *IF*-conditions and various input values.

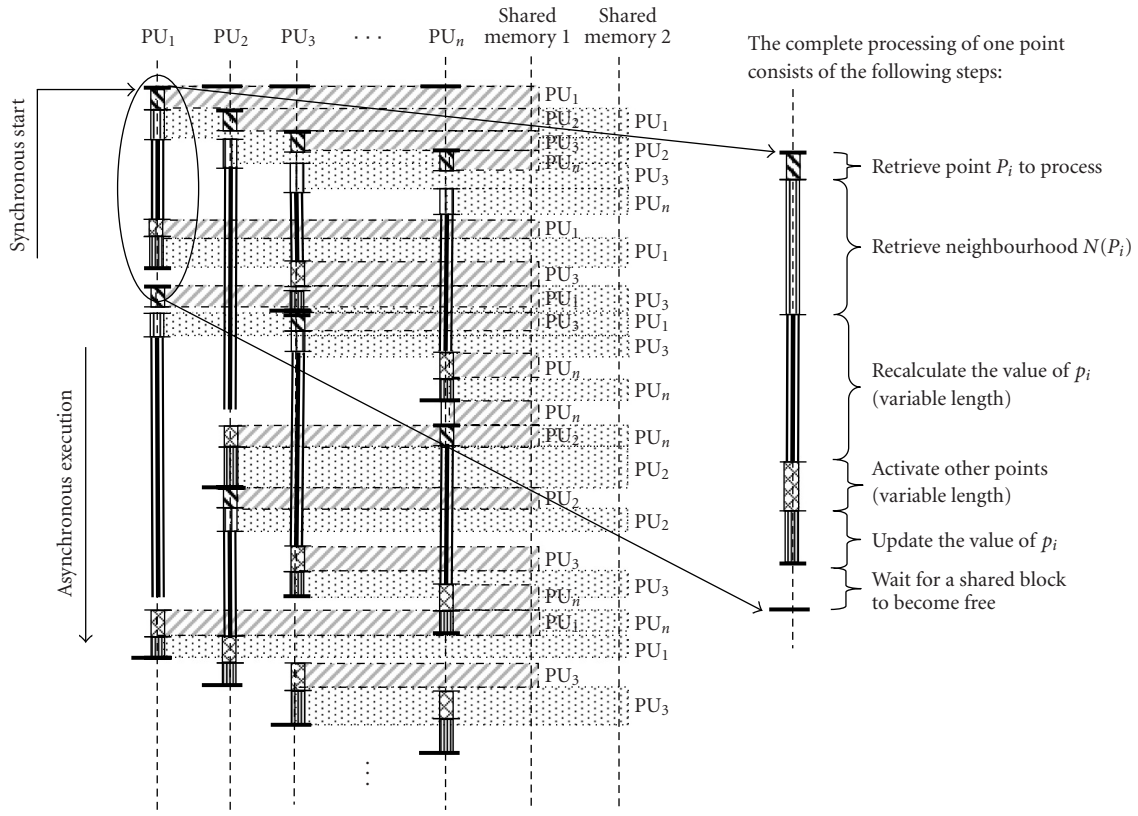


FIGURE 4: Asynchronous execution of the code on four processing units (PU_1 to PU_4) and accesses (in grey) to a shared block.

Whenever the memory is idle, it can (and should) be used to retrieve other data to process. The fact that the algorithms operate locally (using only the information from the neighborhood) characterizes this algorithm family by a fine granularity. On the other hand, the nonlinear functions categorize these algorithms rather into the medium granularity group because, in most cases, a fully functional ALU is necessary to implement the computation. These considerations impose the choice of an MIMD architecture. The processors operate in the SPMD[5] mode, corresponding best to the data flow diagram given by Figure 3.

After a synchronous start of all the processing units, the variable length of some portions of the code gives birth to an asynchronous execution, (cf. Figure 4). The asynchronous execution is advantageous for parallelizable algorithms with numerous *IF*-conditions, because it randomizes the access to the shared blocks. Simultaneous accesses become rare and their HW management is easier. After the analysis of various algorithms, it becomes clear that the choice of asynchronous execution of the code on several PUs is a natural choice for the level family.

Note, that despite the asynchronous execution, the PDE-based algorithms have one or more synchronization points: the end of one iteration. This is indicated by either (i) emptiness of one of the active points memory pages (narrowband-type algorithms), or (ii) end of the raster scan of the image (global scope-type algorithms). The end of the algorithm is indicated by either (i) emptiness of both active points memory pages (for the narrowband-type algorithms), or (ii) the number of necessary iterations (both algorithm types), or (iii) the convergence (both algorithm types).

3. ARCHITECTURE

The image processing domain is known for various algorithm granularity and data dependency. Indeed, the data dependency and granularity are two factors that have major influence on the choice of parallel implementations. Historically, the fundamental model of parallel architectures has been introduced by Flynn [56]. The further effort has been concentrated, besides the computation resources, on the efficiency of the communication configurations (see Cypher and Sanz [57]).

The massive parallelism is efficient for regular algorithms with fine granularity (cf. Gibbons and Rytter [58], Broggi *et al.* [59] or artificial retine by Manzanera [60]). On the other hand, if it used for random memory access implementations, the chip activity versus occupied surface will become poor. The same arguments are valid in the case of SIMD-type architectures (see Cypher and Sanz [57] or e.g., survey in [61]).

[5]Single program multiple data—the processors execute asynchronously the same program.

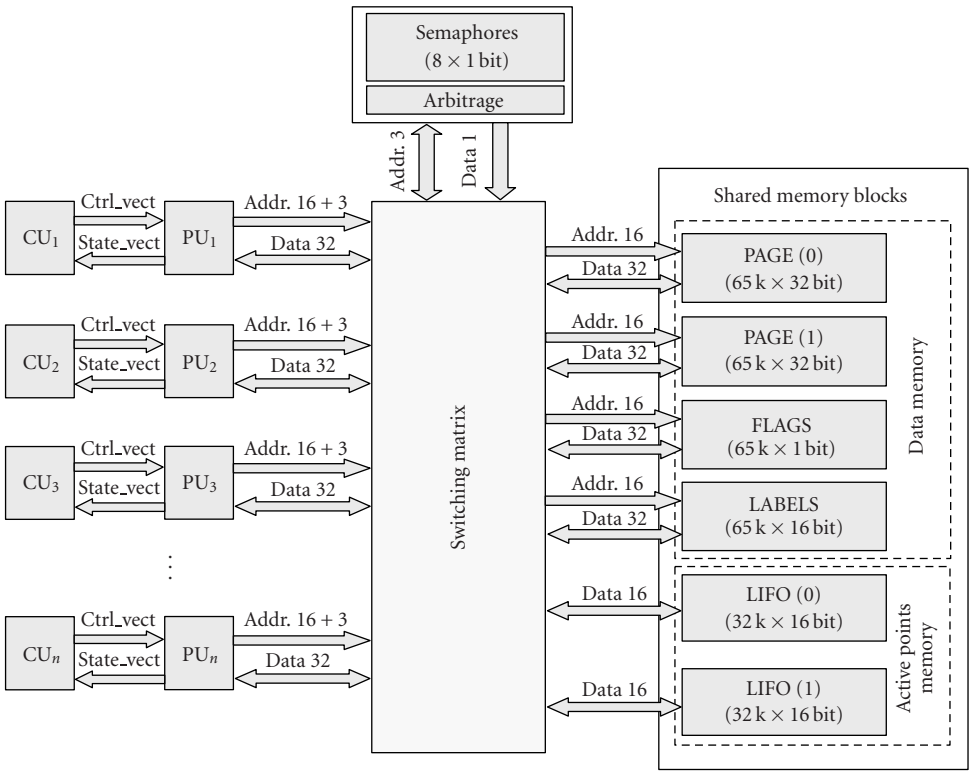


Figure 5: Global overview of the architecture.

Hence, the image analysis community started to consider, as a possible execution platform for mean and high granularity algorithms, in the late 1980s, programmable, multiprocessor, one-chip architectures. See for example [62], [63] or the survey of multiprocessor architectures with shared and distributed memory [64]. For another example and additional references, see a motion estimation on a set of video signal processors by De Greef *et al.* [65], or watershed segmentation in Moga *et al.* [66], Noguet [67], or Bieniek [68].

The data flow of the active contours, analyzed in the previous section, makes them correspond better to "weaker" parallelism models where the design effort concentrates on the task and data dependency decomposition, task scheduling, and efficient management of accessing to shared resources. The architecture template, presented in this section by Figure 5, is derived from the data-flow analysis given by Figure 3. In the following, we detail the description of the individual blocks.

Processing and control units

The computation of the propagation speed $\mathcal{F}$, which is generally a nonlinear function, is a challenge for an efficient implementation. It seems necessary to use a fully functional arithmetic logic unit (ALU).

The processing units were realized in VHDL and HandelC as a model of a RISC processor. They are equipped with a set of registers. The used data word width is 32 bits to store a fixed-point data (24 + 8 the integer and fractional part).

Every processing unit is controlled by a control unit (CU). The execution of the algorithm was simulated by coding the algorithm in HandelC. The advantage of this approach is that the functional model can be replaced by another processor model or by an embedded core available on some FPGAs.

Switching matrix

The medium granularity combined with intensive random accesses to the data memory shows that no optimum *fixed* interconnection network can be found for the level set algorithm family. Rather than using a fixed network, one can use a switching matrix which, coupled with semaphores and arbitrage, permits to any processing unit access to any shared block, provided that it is not currently being used by another processing unit. Several PUs can access simultaneously to different shared blocks.

The address buses are 16 + 3 bits (16 bits for the data addressing and 3 bits to address eight semaphores for the following eight shared blocks), four data memory blocks (data PAGE(0), PAGE(1), FLAGS, and LABELS). The active points memory is divided into two blocks, each equipped with a bidirectional reading/writing channel since each of the stacks is in one iteration either read or written but not both.

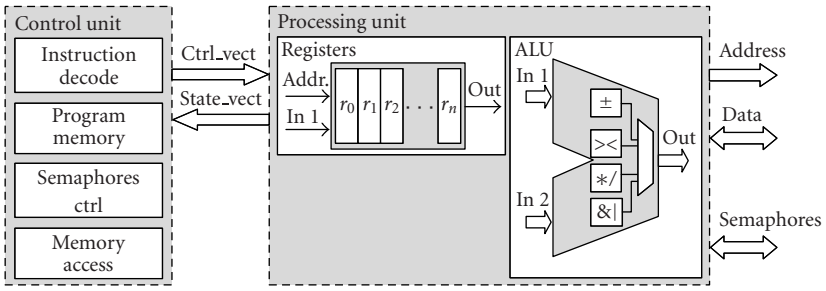


Figure 6: Internal architecture of the processing and control units.

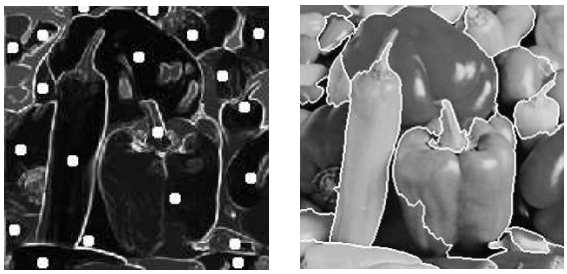


Figure 7: The "peppers" image: (a) gradient and manually placed markers; (b) continuous watershed obtained with massive marching on four processing units.

Semaphores and arbitrage

Every operation asking to access to a shared block uses the semaphores (block semaphores ctrl at Figure 6). The code that performs the semaphore-controlled access must respect the following:

```
loop :
test semaphore x          // test and lock immediately if free
if x is not free jump loop // repeat otherwise
read/write                // access to the memory
release semaphore x       // release the semaphore
```

Whenever a semaphore is tested, it is (by the same instruction) immediately locked, provided that it was free. If not, the test is repeated as long as the semaphore can be allocated to the asking processor. After the reading/writing, the semaphore is released. The semaphores are invisible to the user provided that the compiler generates the corresponding code.

Whenever a simultaneous access to a shared block occurs, an arbitrage is used to prevent conflicts. The arbitrage is a standard block that makes part of most modern multi-processor platforms. The ideal arbitrage, usually done on the *first-come-first-served* basis, and often realized as a finite state machine, is quite costly in terms of the silicium surface. We can benefit from the randomness of the asynchronous execution, limiting the likelihood of simultaneous accesses, and saving the space by using a simple arbitrage assigning the processors an uneven priority. Obviously, this is only possible up to a certain number of processors however. In this paper, we have evaluated the feasibility by measuring the activity up to four processors (see Figure 8 showing the activity distribution).

Data memory

A low data dependency that characterizes the level set family algorithms permits to use a simple global shared memory management, being referred to in the literature as *weak consistency* model, introduced in [69]. The weak consistency is characterized by three conditions (cf. [70]). (i) Before a READ or WRITE access for any processor is allowed, all synchronizations must be achieved. (ii) Before a synchronization access is allowed, all previous READ or WRITE accesses must be achieved. (iii) Synchronization accesses are sequentially consistent with respect to each other. Note that no condition concerns the order in which the accesses are performed. See [70] for details and comparison with other consistency models.

The synchronization points are imposed by the iterative nature of the algorithms. All active points must be processed (in arbitrary order) in one iteration, before the following iteration can start. This is ensured on this architecture by the fact that the data to process are read from one memory page, and the results are written to the other. As soon as all the points in one iteration are processed (all READ and WRITE accesses are achieved), the roles of the pages PAGEs(i), i = 0,1, switch. Switching the roles of the memory pages represents the synchronization.

This architecture is conceived as scalable. According to the computational power required by a given application, one can use more or fewer processing units. It follows from Figure 3 that the highest data traffic concentrates on the shared memory blocks. Thanks to the nature of the code, the reading and writing directions on both data and active points memory blocks are separated into two one-directional channels. The results of the previous iteration (values u^{n-1}) are read from one page and the new values (u^n) are written to the other. This corresponds perfectly to the weak consistency model.

Active points memory

The READ and WRITE accesses to perform on the image data are controlled by data stored in the active points memory. It is organized in two pages. One page contains points

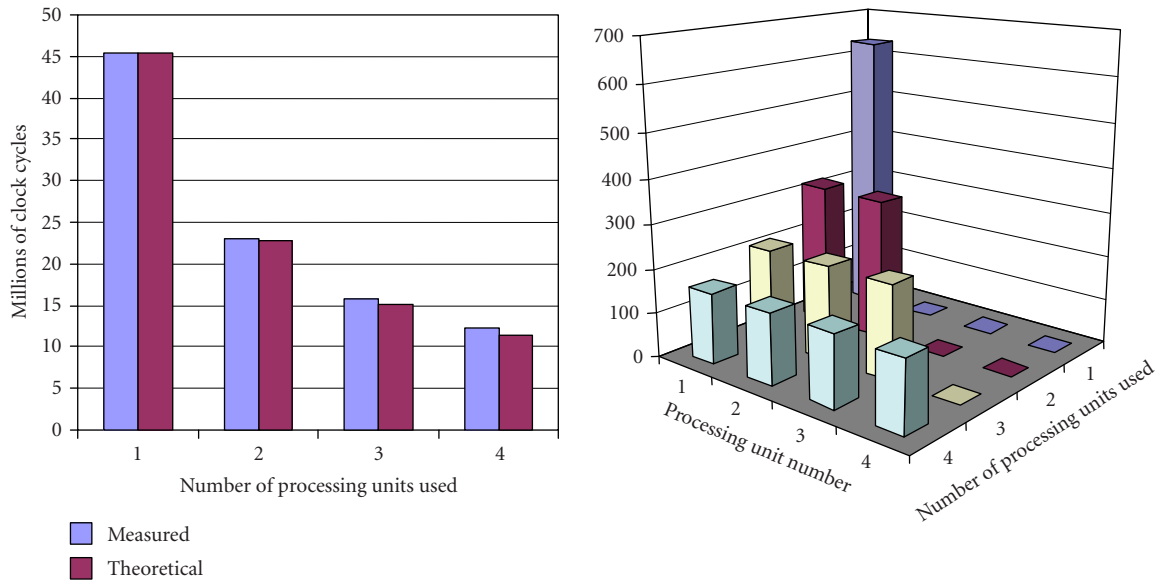


FIGURE 8: (a) The execution time of the algorithm in function of the number of parallely working processing units. (b) The activity load distributed over several processing units, number of points processed by every processing unit (X thousands).

to process in the current iteration. This page is progressively emptied as the data are processed. The second page contains points to process in the next iteration. It is progressively fed with data. The emptiness of one page represents the synchronization point. The roles of the pages (for both data and active points memory) switch. The emptiness of both active points pages represents the end of the algorithm.

The reading/writing direction to the data and active points memory blocks is controlled by using a boolean variable *switch* which commutes at the end of every iteration. For the sake of universality, it is left to the programmer's responsibility to control the reading.

Thanks to the fact that the processing order is indifferent, this memory can be implemented by using two LIFOs. Compared to a FIFO, using LIFO eliminates the transport delay.

For most applications, the reading should always be done on data page(*switch*) and LIFO(*switch*) and writing on page(*switch*) and LIFO(*switch*). The binary *switch* value can be derived from the zero bit of the iteration number n.

Flags

The labels and flags are similar to the data memory with a smaller word size. The labels and flags are available to the programmer for an additional algorithm control and region propagation.

4. PERFORMANCE EVALUATION

The performance of this architecture has been tested by running two different types of PDE-based algorithms: a continuous watershed and an object-tracking application.

The objective of the watershed computation is to justify the choice to use an MIMD architecture by testing whether the overall computational effort is uniformly distributed over all the processors used. The objective of the tracking application (cf. Section 4.2), is to evaluate the overall bandwidth of the architecture, and the capability to run a computationally expensive application in real time.

4.1. Evaluation test 1: A continuous watershed implementation

Recall that, in terms of PDEs, watersheds can be obtained by calculating a weighted distance function to a given set of sources, corresponding to the markers [34], while propagating simultaneously the labels

$$\|\nabla u(x,y)\| = \frac{1}{\|\nabla I\|}. \tag{1}$$

Recall that the set of sources must be identical with the set of local minima in the image, as shown in [71]. The distance function was computed in a semiparallel way, on four parallely operating processing units, from a manually placed set of markers, see Figure 7.

Figure 8b shows the execution time (in terms of total clock cycles against the number N of processing units operating in parallel). The obtained number of clock cycles corresponds to the theoretical number of clock cycles calculated as $\text{clk}_N = \text{clk}_1/N$. The measured execution time (expressed in terms of clock cycles) slightly exceeds the theoretical value because of the access to the shared blocks (memory, LIFO), controlled by a semaphore. Figure **??** gives the computational load distributed over the processing units in function of the number of processing units used. The computational load is expressed in terms of number of points processed by every processing unit. If only one unit is used, the total computational load is covered by this unit. If more processing units operate in parallel, the load is uniformly distributed.

Table 1: The obtained bandwidth for watershed computation versus other platforms.

Platform	Frequency	Bandwidth (10^3points/s)
Proposed MIMD architecture Four RISC processors	120 MHz FPGA	2 610
PC with P4 Win 2000	1.6 GHz	827
IPAQ with Xscale WinCE	400 MHz	120
IPAQ with Strong ARM WinCE	200 MHz	50

Table 1 compares the bandwidth of weighted distance computation with simultaneous propagation of source labels, obtained by using massive marching implemented on various platforms. The bandwidth is computed as the number of points in the image divided by the execution time. The execution time of the proposed MIMD architecture was obtained by counting the clock cycles during simulation (HandelC code). The execution time obtained on a PC/P4, IPAQ/Xscale and StrongARM corresponds to the processor time spent in the process (programmed in C).

Note that the bandwidth of every given architecture is somewhat lesser than the theoretical bandwidth because some points are activated several times. The computation complexity of massive marching is roughly $\mathcal{O}(N)$, with N being the number of points in the image. It exceeds N by the number of reactivated points because of using a nonequidistant propagation front.

4.2. Evaluation test 2: Object-tracking application

To test the performance of this architecture, we use a model-free, gradient-based object-tracking algorithm proposed in [72].

4.2.1. A gradient-based attraction field

Consider an image $\mathcal{I}$ and some gradient of $\mathcal{I}$, $g = \nabla\mathcal{I}$. Let

$$g_K = g * K, \tag{2}$$

where K is some triangular window $\mathbb{Z}^2 \to \mathbb{R}^+$, such that

$$K(x,y) = \begin{cases} 1 - \alpha\left(x^2+y^2\right)^{1/2} & \text{if } \left(x^2+y^2\right)^{1/2} < \dfrac{1}{\alpha}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that in the signal processing domain, convoluting with such a window is a frequency filter. However, filtering is not the objective here.

∇g_K represents a gradient-dependent integrator with interesting properties. Generally, the evolution of a curve $\mathcal{C}$ writes

$$\frac{\partial \mathcal{C}}{\partial t} = \mathcal{F}\vec{n}, \tag{4}$$

where $\vec{n}$ is the normal vector to $\mathcal{C}$, and $\mathcal{F}$ represents the motion speed. For the contour-based tracking, we propose

$$\mathcal{F} = \nabla g_K. \tag{5}$$

It can be shown (by approximating g in (2) by a Dirac impulse δ, and computing $\mathcal{F}$ in (5) in a discrete form) that ∇g_K is a bidirectional integrator pointing towards the crest of the gradient g from both sides.

The advantage of using a bidirectional integrator is twofold: (i) it allows the contour to converge towards the gradient maximum from both sides, and (ii) it eliminates the necessity to use a constant one-directional attraction force there, where the data is zero. This fact eliminates the problem of local breaches in the gradient, often introducing leakage in object reconstruction. Attempts to alleviate this problem were made in [73] introducing a viscous watershed capable to slow down the propagation in such narrow openings. Although the leakage could probably be alleviated by using curvature, the leakage problem does not occur when using ∇g_K, since on zero gradient the contour does not move.

Let ϕ represent some feature of the object to track. Supposing that this feature is unstable in time, or perturbed by external phenomena, one may need to employ an additional cue to enhance the stability. Natural gesture speed is one of the possible cues to track individuals. This fact is also used in defining the capture range of the contours. Suppose that the maximum interframe displacement of the object is bounded by D. This information should be taken into account by letting $\operatorname{supp}\{(x,y) \mid K(x,y) > 0\}$ be a circle of radius D, generating a nonzero attraction field in a narrow zone around the contour. Hence, a convenient value of α in (3) is $a = 1/D$.

Indeed, as the attraction force stops on the zero crossing of the gradient, its principle is similar to the Haralick [74] edge detector, which detects edges on zero crossing of the second derivative of $\mathcal{I}$ in the gradient direction. Kimmel and Bruckstein in [75] reformulate the Haralick edge detector in terms of the level set framework and shows how it can be combined with additive constraints to segment images. As stated before, our objective is the contour-based object tracking. Whereas various motion predictors can be used to predict the displacement direction according to the past, arbitrary deformations of the object give birth to a displacement field with locally varying direction. Any contour-based tracking must therefore be able to handle both partially forward and backward displacements of the contour. A good overview of other existing attraction vector fields can be found in [76].

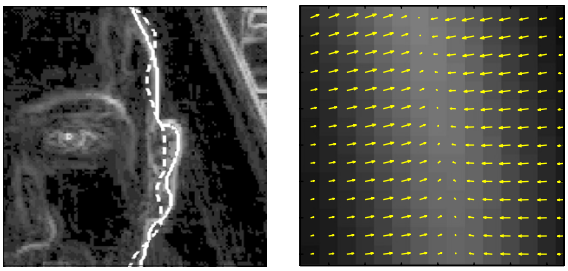


Figure 9: (a) The initial (dashed) and final (solid line) position of the contour, and (b) zoom on the attraction force field F.

4.2.2. Application

By integrating (4), the current contour C^n of the object is obtained by using the attraction field g_K^n generated by the current frame $\mathcal{I}^n$, and the contour C^{n-1} in the previous frame (cf. Figure 9):

$$C^n = \lim_{T\to\infty} \int_0^T \nabla g_K^n(C)\vec{n}\, dt + C^{n-1}, \tag{6}$$

$$\text{with } C(t=0) = C^{n-1}, \tag{7}$$

where $g_K^n = g * K$,

$$g(p) = \frac{\nabla_{\text{Lab}} I(p)}{1 + d_{\Omega|\phi}(I(p))}. \tag{8}$$

The ∇_{Lab} denotes the gradient on the Lab colour space. The particularity of the Lab space is that it is perceptually uniform, and ∇_{Lab} is locally Euclidean. The $d_{\Omega|\phi}$ denotes the distance to a given feature. We use a feature based on the skin chroma. We take $\Omega' \equiv$ HLS, and $\phi = \{x \in \text{HLS} | x_H \in [-20^o, 50^o]\}$. This feature is only related to hue, thus the distance $d_{\text{HLS}|\phi}$ is the angular distance d^α to the skin chroma ϕ. The size of the triangular window K is ten pixels, that is, $\alpha = 0.1$, calculated from a natural gesture speed as seen by our camera.

Initialization

The description of the initialization of the tracking is outside the scope of this paper. It can be successfully done by combining several features, see for example [77], using the face colour and shape or [78] combining the colour and motion (in a car application, no perturbing motion is present in the background before the car runs).

4.2.3. Implementation

In the following, we outline the details concerning the implementation of the object tracking on the proposed architecture.

This architecture has been simulated using the HandelC programming language. The control units have been replaced by a pipelined model controlling each processing unit, equipped with a fully functional ALU realizing the basic arithmetic/logic operations in fixed-point precision, and equipped with a set of registers. The algorithms have been hardcoded in the control units in HandelC instructions. Note that every HandelC instruction is executed in one clock cycle.

Table 2: Frame parameters.

Frame size (X × Y)	324 × 428
Number of points in the frame	138 672
Frames per second	15
Data flow (points per second)	2 080 080

Application parameters

The video stream contains 15 frames per second, each 324 × 428 pixels, giving total data flow $2.08 \cdot 10^6$ pixels per second (cf. Table 2).

The narrowband width has been set to 20 points (ten to each side of the contour) and the mean length of the contour of the face (cf. Figure 10) to track is approximately 600 points, giving in average 12 000 active points to update per iteration, see Table 3.

The above given face tracking application requires 25 iterations in every frame for the contour to adapt itself to the new position of the face. (We consider that natural gesture speed, camera resolution, and distance to the face limit the interframe displacement of the drivers face to approximately 10 pixels.) Every five iterations, the narrowband needs to be reinitialized (cf. Table 4).

Instruction count for various algorithm steps

The construction of the attraction force field requires one convolution (cf. (2)). An $N \times N$ fast 2D convolution can be efficiently implemented by a serie of $2N$ 1D FFT applied to the columns and rows, N^2 multiplications, and a series of $2N$ 1D IFFT. Efficient algorithms exist to perform FFT/IFFT in place, see for example [79], and modern DSPs are equipped with efficient, highly optimized blocks calculating fast the FFT, for example [80].

We suppose that the convolution is computed on a companion chip. In the following, we focus on the implementation of the level-set-based part of the application, that is, the (i) initialization and construction of the narrowband, (ii) contour evolution.

The gradient can be calculated with two additions and two divisions (if central differences are used). The attraction force ∇g_K calculated on the entire frame requires 277, 344 additions and as many multiplications (cf. Table 5). The construction of the narrowband, by using massive marching, requires two steps: (i) the *interpolation* to initialize the contour can be done with 4 additions per point and (ii) the *propagation* of the distance function requires 5 additions and 6 multiplications per point. Performed twice (Jacobi and Gauss-Seidel steps) on 12 000 points (narrowband size from Table 3) gives 216 000 additions and 144 000 multiplication required to construct the narrowband. The narrowband is reconstructed five times per frame, giving the level set inherent computational effort of 1 080 000 additions and 720 000 multiplications per image frame.

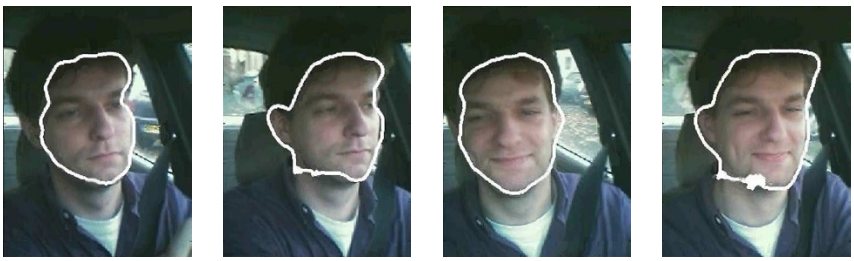


Figure 10: Contour tracking applied to driver's face extraction, using the weighted gradient (skin chroma being the feature of interest). Randomly chosen images from a video sequence.

Table 3: Narrowband parameters.

Narrow bandwidth (points)	20
Approximate mean contour length (points)	600
Number of points in the narrowband	12 000

Table 4: Object-tracking application parameters.

Number of iterations before reinitialization	5
Reinitializations per frame	5
Number of iterations per frame	25

The actual curve evolution involves several steps: (i) the evolution speed $\mathcal{F}$ requires 3 addition and 4 multiplications (including the gradient of the distance function U), (ii) the integration is done in one additions and one multiplication, giving in total 4 additions and 5 multiplications per point. Multiplied by 12 000 points in the narrowband (48 000 additions and 60 000 multiplications) and by 25 iterations per frame gives $1.2 \cdot 10^6$ additions and $1.5 \cdot 10^6$ multiplications per frame. The total application effort is $2.56 \cdot 10^6$ additions and $2.50 \cdot 10^6$ multiplications per frame, representing in total 75.8 MFLOPS to run in real time.

Table 6 presents the lower limits of the bandwidth obtained for different steps of the object-tracking application. The computation of the gradients ∇g_K and ∇u requires the same elementary operations (differences and extrema computation on the neighborhood), and presents obviously the same bandwidth $19.3 \cdot 10^6$. The limiting factor in this case is the neighborhood extraction from the input image. We have obtained the same bandwidth estimation for the integration step. The integration does not read the neighborhood (already stored in the registers) but only writes the integration result. Its performance can sometimes be limited by the bandwidth of the foregoing step.

The bandwidth $2.61 \cdot 10^6$ points/s, obtained for the narrowband construction, includes the detection of the initial contour position by interpolation and the propagation of the distance function.

We evaluate the performance of the architecture by computing the processing time of the each algorithm stage as a function of the number of processed points and these measured worst-case bandwidths. The processing time of all the steps is obtained by multiplying the worst-case bandwidth, the number of iterations, and the number of the points to process.

The sum of the processing times of individual steps gives the frame-to-frame processing time $6.18 \cdot 10^{-2}$ seconds, corresponding to 16.3 processed frames per second.

The performance, outlined in Table 7, compares the execution time of one iteration of the above-detailed object-tracking application on this architecture compared to similar results obtained on other platforms reported in the literature.

The nVIDIA GeForce2 graphic card, see [16], operates in integer accuracy, and is therefore less useful for algorithms requiring multiple iterations. The application running on PC P4, see [81], was implemented by using the additive operator splitting (AOS) scheme, permitting greater integration step, and requiring thus fewer iterations.

4.3. Power assessment

As the silicium surface on FPGAs continues to grow (to become comparable to ASICs), the computational power is no longer a limiting factor for the design. Instead, the preoccupations concern more and more the energy dissipation and the system autonomy.

The energy budget of some algorithm can be characterized by the energy necessary to execute the elementary operation multiplied by the number this operation is executed. Suppose that this algorithm is to be executed in a limited time. A parallel execution (provided that the algorithm is parallelizable) will allow to reduce the clock frequency (compared to the clock frequency of the sequential implementation) and reduce the energy budget of the elementary operation.

Though it is important to take into account the energy considerations as soon as possible during the design, at this development stage, it is still difficult to estimate precisely the power consumption. The execution of the algorithms was simulated by using a general-purpose RISC processor model. The power consumption was then estimated by using the consumption reported by various soft-core processors manufacturers: for Microblazer (Xilinx), see [82]; for ARM 9 family see [83]; and compared with typical-to-maximum thermal dissipation reported for Pentium 4 at 1.6 GHz (see [84]), compare Table 8.

Table 5: Instruction count for various steps.

Instruction count for various steps	*Additions*	*Multiplications*
Preprocessing		
∇g_K (operations per point)	2	2
Total per frame (additions, multiplication)	277 344	277 344
Construction of the narrowband		
Interpolation (additions, multiplications per point)	4	0
Propagation (additions, multiplications per point)	5	6
Total per initialization (additions, multiplication)	216 000	144 000
Total level-set-inherent computational effort	1 080 000	720 000
Curve evolution		
Evolution speed $\mathcal{F} = \nabla g_K \cdot \nabla U$	3	4
Integration (additions, multiplications per point) $U = U - (\mathcal{F}\, dt)$	1	1
Curve evolution per point (additions, multiplications)	4	5
Curve evolution per iteration (additions, multiplication)	48 000	60 000
Total curve evolution per frame	1 200 000	1 500 000
Total application per frame (curve evolution + level set inherent)	2 557 344	2 497 344
Overall real-time computational effort (FLOPS)	$75.8 \cdot 10^6$	

Table 6: The Execution Time of the Object Tracking Application.

Algorithm step	Estimated bandwidth (point/s)	Number of iterations	Number of points	Processing time (s)
Initialization				
Gradient ∇g_K	$19.3 \cdot 10^6$	1	138 672	$7.19 \cdot 10^{-3}$
Narrowband construction	$2.61 \cdot 10^6$	5	12 000	$2.30 \cdot 10^{-2}$
Evolution				
Gradient ∇u	$19.3 \cdot 10^6$	25	12 000	$1.56 \cdot 10^{-2}$
Integration u^{n+1}	$19.3 \cdot 10^6$	25	12 000	$1.56 \cdot 10^{-2}$
Total execution time (per frame)				$6.13 \cdot 10^{-2}$
Application frame processing rate (frame/s)				16.3

Table 7: The execution time of one iteration, compared to similar algorithms on other platforms.

Platform	Frequency	Execution time for one iteration (ms)
Proposed MIMD architecture Four RISC processors	120 MHz/FPGA	1.25
Graphic hardware nVIDIA GeForce2	250 MHz	4
PC with P4/Win 2000	1.6 GHz	19.1

Table 8: Comparison of power consumption.

Processor	Power consumption (W)
Microblazer / Xilinx	0.11
ARM9 / ARM	0.14
Pentium4 (1.6 GHz)/ Intel	60–75

5. CONCLUSIONS

In this paper, we present an embedded architecture for real-time image processing using level-set-based active contours. The contribution of this paper is twofold. In its first part, the text proposes a unifying insight into the level set framework from the system design point of view, to propose a unique iteration type with two different types of memory access: random memory access and sequential memory access. Then it analyzes the data flow to define, in the second part of the text, a scalable architecture fitting the real-time needs and taking into account the limited energy autonomy of embedded platforms and the silicium surface on commercially available FPGAs.

The performance of the proposed architecture has been studied on two benchmarks.

The first one, computation of a weighted distance transform with simultaneous propagation of region labels, is to verify the uniformity of the data flow and the distribution of the computational burden over all the processing units. This benchmark compares the real execution time against the theoretical execution time (obtained as the time needed by one processing unit divided by the number of processing units

operating in parallel). The results show a linear increase of performance and a balanced activity at least up to four independently operating processing units.

The second benchmark implements an active-contour-based object-tracking algorithm. The purpose of this test is to evaluate the capability of this platform to run in real-time applications with intensive random memory accesses. Section 4.2.3 lists the details concerning the computational complexity of the application in terms of number of elementary operations. The simulation results show that the above-presented contour tracking application can be run on this architecture in real time, provided that the processors are clocked at 120 MHz, and one instruction executes in one clock cycle. Hence, the architecture specifications made in the first part of the text are confirmed.

The scalability of this architecture consists in replicating the processing units. Physically, their number is limited by the silicium available on the chip; and logically, by the data-flow balance on all the blocks of the architecture. A time-costly computation will allow a linear increase of the performance up to a higher number of processing units, before the busses and the memory blocks saturate. From Figure 3, it follows that the highest data flow concentrates on the READ data memory. Although it has not been used in this paper, two possible improvements will make the data flow on the individual memory blocks more uniform: (i) the entire four-neighborhood can be retrieved in one clock cycle by using another memory organization, as proposed by Noguet in [67], or (ii) the READ data memory flow can be divided by two by using a dual-port memory for the data memory pages. However, both options will lead to some increase of complexity of the switch.

REFERENCES

[1] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

[2] S. Osher and R. P. Fedkiw, "Level set methods: an overview and some recent results," *Journal of Computational Physics*, vol. 169, no. 2, pp. 463–502, 2001.

[3] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Science*, Cambridge University Press, Cambridge, UK, 1996.

[4] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press, New York, NY, USA, 2001.

[5] D. Adalsteinsson and J. A. Sethian, "A fast level set method for propagating interfaces," *Journal of Computational Physics*, vol. 118, no. 2, pp. 269–277, 1995.

[6] R. Malladi, J. A. Sethian, and B. C. Vemuri, "A fast level set based algorithm for topology-independent shape modeling," *Journal of Mathematical Imaging and Vision*, vol. 6, no. 2-3, pp. 269–289, 1996.

[7] F. Precioso and M. Barlaud, "B-spline active contour with handling of topology changes for fast video segmentation," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 6, pp. 555–560, 2002, Special Issue on Image Analysis for Multimedia Interactive.

[8] G. Cserey, C. Rekeczky, and P. Földesy, "PDE-based histogram modification with embedded morphological processing of the level-sets," *Journal of Circuits, Systems and Computers*, vol. 12, no. 4, pp. 519–538, 2003.

[9] J. Weickert, B. M. T. H. Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering," *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 398–410, 1998.

[10] F. Catté, P.-L. Lions, J.-M. Morel, and T. Coll, "Image selective smoothing and edge detection by nonlinear diffusion," *SIAM Journal on Numerical Analysis*, vol. 29, no. 1, pp. 182–193, 1992.

[11] R. Goldenberg, R. Kimmel, E. Rivlin, and M. Rudzsky, "Fast geodesic active contours," *IEEE Trans. Image Processing*, vol. 10, no. 10, pp. 1467–1475, 2001.

[12] P. Smereka, "Semi-implicit level set methods for curvature and surface diffusion motion," *Journal of Scientific Computing*, vol. 19, no. 1-3, pp. 439–456, 2003.

[13] S. Holmgren and D. Wallin, *Performance of High-Accuracy PDE Solvers on a Self-Optimizing NUMA Architecture*, vol. 2150 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2001.

[14] J. A. Sethian, "Parallel level set methods for propagating interfaces on the connection machine," Department of Mathematics, University of California at Berkeley, Berkeley, Calif, USA, 1989.

[15] M. Rumpf and R. Strzodka, "Nonlinear diffusion in graphics hardware," in *Proc. EG/IEEE TCVG Symposium on Visualization (VisSym '01)*, pp. 75–84, Ascona, Switzerland, May 2001.

[16] M. Rumpf and R. Strzodka, "Level set segmentation in graphics hardware," in *Proc. International Conference on Image Processing (ICIP '01)*, vol. 3, pp. 1103–1106, Thessaloniki, Greece, October 2001.

[17] J. E. Cates, A. E. Lefohn, and R. T. Whitaker, "GIST: an interactive, GPU-based level set segmentation tool for 3D medical images," *Medical Image Analysis*, vol. 8, no. 3, pp. 217–231, 2004.

[18] C. Sigg, R. Peikert, and M. Gross, "Signed distance transform using graphics hardware," in *Proc. 14th IEEE Visualization Conference (VIS '03)*, pp. 83–90, Seattle, Wash, USA, October 2003.

[19] K. Hwang, P. S. Tseng, and D. Kim, "An orthogonal multiprocessor for parallel scientific computations," *IEEE Trans. Comput.*, vol. 38, no. 1, pp. 47–61, 1989.

[20] T. Gijbels, P. Six, L. Van Gool, F. Catthoor, H. De Man, and A. Oosterlinck, "A VLSI-architecture for parallel non-linear diffusion with applications in vision," in *Proc. IEEE Workshop on VLSI Signal Processing VII*, pp. 398–407, La Jolla, Calif, USA, October 1994.

[21] E. Dejnožková and P. Dokládal, "A parallel architecture for curve-evolution PDEs," *Image Analysis and Stereology*, vol. 22, pp. 121–132, 2003.

[22] R. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, K. L. Pocek and J. Arnold, Eds., pp. 126–135, IEEE Computer Society, Napa Valley, Calif, USA, April 1996.

[23] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Proc. 27th International Symposium on Computer Architecture*, pp. 225–235, British Columbia, Canada, 2000.

[24] Y. Li, T. Callahan, E. Dernell, R. Harr, U. Kurkure, and J. Stockwood, "Hardware-software co-design of embedded reconfigurable architectures," in *Proc. 37th Design Automation Conference (DAC '00)*, pp. 507–512, Los Angeles, Calif, USA, June 2000.

[25] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 7, pp. 629–639, 1990.

[26] L. Alvarez, P.-L. Lions, and J.-M. Morel, "Image selective smoothing and edge detection by nonlinear diffusion. II," *SIAM Journal on Numerical Analysis*, vol. 29, no. 3, pp. 845–866, 1992.

[27] S. Schüpp, *Prétraitement et segmentation d'images par mise en oeuvre de techniques basées sur les équations aux dérivées partielles : application en imagerie microscopique biomedicale*, Ph.D. thesis, Université de Caen Basse-Normandie, Caen Cedex, France, December 2000.

[28] B. Kimia and K. Siddiqi, "Geometric heat equation and nonlinear diffusion of shapes and images," *Computer Vision and Image Understanding*, vol. 64, no. 3, pp. 305–322, 1996.

[29] T. Lindeberg, *Scale-Space Theory in Computer Vision*, Kluwer Academic, Dordrecht, The Netherlands, 1994.

[30] L. Alvarez, F. Guichard, P.-L. Lions, and J.-M. Morel, "Axioms and fundamental equations of image processing," *Archive for Rational Mechanics and Analysis*, vol. 123, pp. 199–257, 1993.

[31] R. Brockett and P. Maragos, "Evolution equations for continuous-scale morphology," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP '92)*, vol. 3, pp. 125–128, San Francisco, Calif, USA, March 1992.

[32] R. van den Boomgaard, *Mathematical morphology: extensions towards computer vision*, Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands, March 1992.

[33] F. Meyer and P. Maragos, "Nonlinear scale-space representation with morphological levelings," *Journal of Visual Communication and Image Representation*, vol. 11, no. 2, pp. 245–265, 2000.

[34] L. Najman and M. Schmitt, "Watershed of a continuous function," *Signal Processing*, vol. 38, no. 1, pp. 99–112, 1994.

[35] A. Montanvert and J. M. Chassery, *Géométrie discrète en analyse d'images*, Hermès, Paris, France, 1991.

[36] R. Kimmel, *Curve evolution on surfaces*, Ph.D. thesis, Technion - Israel Institute of Technology, Haifa, Israel, May 1995.

[37] J. A. Sethian, "A marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[38] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1987.

[39] D. Terzopoulos, A. Witkin, and M. Kass, "Constraints on deformable models: recovering 3D shape and nonrigid motions," *Artificial Intelligence*, vol. 36, no. 1, pp. 91–123, 1988.

[40] L. Cohen, "On active contour models and balloons," *Computer Vision, Graphics, and Image Processing*, vol. 53, no. 2, pp. 211–218, 1991.

[41] V. Caselles, F. Catté, T. Coll, and F. Dibos, "A geometric model for active contours in image processing," *Numerische Mathematik*, vol. 66, no. 1, pp. 1–31, 1993.

[42] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Topology independent shape modeling scheme," in *Geometric Methods in Computer Vision II*, vol. 2031 of *Proceedings of SPIE*, pp. 246–258, San Diego, Calif, USA, July 1993.

[43] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Evolutionary fronts for topology-independent shape modeling and recovery," in *Proc. 3rd European Conference on Computer Vision (ECCV '94)*, vol. 800 of *Lecture Notes in Computer Science*, pp. 3–13, Stockholm, Sweden, May 1994.

[44] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: a level set approach," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, no. 2, pp. 158–175, 1995.

[45] R. Malladi, R. Kimmel, D. Adalsteinsson, G. Sapiro, V. Caselles, and J. A. Sethian, "A geometric approach to segmentation and analysis of 3d medical images," in *Proc. Mathematical Methods in Biomedical Image Analysis Workshop (MMBIA '96)*, San Francisco, Calif, USA, June 1996.

[46] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," in *Proc. 5th IEEE International Conference on Computer Vision (ICCV '95)*, pp. 694–699, Cambridge, Mass, USA, June 1995.

[47] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Gradient flows and geometric active contours models," in *Proc. 5th International Conference on Computer Vision (ICCV '95)*, pp. 810–815, Cambridge, Mass, USA, June 1995.

[48] R. Malladi and J. A. Sethian, "Image processing: flows under min/max curvature and mean curvature," *Graphical Models and Image Processing*, vol. 58, no. 2, pp. 127–141, 1996.

[49] S. Jehan-Besson, M. Gastaud, M. Barlaud, and G. Aubert, "Region-based active contours using geometrical and statistical features for image segmentation," in *Proc. IEEE International Conference in Image Processing (ICIP '03)*, vol. 2, pp. 643–646, Barcelona, Spain, September 2003.

[50] L. Alvarez, J. Weickert, and J. Sánchez, "A scale-space approach to nonlocal optical flow calculations," in *Proc. 2nd International Conference on Scale-Space Theories in Computer Vision (Scale-Space '99)*, pp. 235–246, Corfu, Greece, September 1999.

[51] J. B. T. M. Roerdink and A. Meijster, "The watershed transform: definitions, algorithms and parallelization strategies," *Fundamenta Informaticae*, vol. 41, no. 1-2, pp. 187–228, 2000.

[52] E. Dejnožková, *Architecture dédiée au traitement d'image basé sur les équations aux dérives partielles*, Ph.D. thesis, Ecole Nationale Supérieure des Mines de Paris, Paris, France, March 2004.

[53] J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.

[54] J. A. Sethian, "Level set methods and fast marching methods," Tech. Rep., Department of Mathematics, University of California, Berkeley, Calif, USA, 1999. http://math.berkeley.edu/~sethian/level_set.html.

[55] S. Kim, "An $\mathcal{O}(N)$ level set method for eikonal equations," *SIAM Journal on Scientific Computing*, vol. 22, no. 6, pp. 2178–2193, 2001.

[56] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[57] R. Cypher and J. L. C. Sanz, "SIMD architecture and algorithms for image processing and computer vision," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 12, pp. 2158–2174, 1989.

[58] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, UK, 1988.

[59] A. Broggi, G. Conte, F. Gregoretti, C. Sansoè, and L. M. Reyneri, "The Paprica massively parallel processor," in *Proc. 1st IEEE International Conference on Massively Parallel Computing Systems (MPCS '94)*, pp. 16–30, Ischia, Italy, May 1994.

[60] A. Manzanera, "Morphological segmentation on the programmable retina: towards mixed synchronous/asynchronous algorithms," in *Proc. 6th International Symposium on Mathematical Morphology (ISMM '02)*, H. Talbot and R. Beare, Eds., pp. 389–399, Sydney, Australia, April 2002.

[61] T. Le, W. Snelgrove, and S. Panchanathan, "SIMD processor arrays for image and video processing: a review," in *Multimedia Hardware Architectures*, vol. 3311 of *Proceedings of SPIE*, pp. 30–41, San Jose, Calif, USA, January 1998.

[62] M. Maruyama, H. Nakahira, T. Araki, et al., "A 200 MIPS image signal multiprocessor on a single chip," in *Proc. 37th IEEE International Solid-State Circuits Conference (ISSCC '90)*, pp. 122–123, San Francisco, Calif, USA, February 1990.

[63] T. Minami, R. Kasai, H. Yamauchi, Y. Tashiro, J. Takahashi, and S. Date, "A 300-MOPS video signal processor with a parallel architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 12, pp. 1868–1875, 1991.
[64] R. Duncan, "A survey of parallel computer architectures," *IEEE Computer*, vol. 23, no. 2, pp. 5–16, 1990.
[65] E. De Greef, F. Catthoor, and H. De Man, "Mapping real-time motion estimation type algorithms to memory efficient, programmable multi-processor architectures," *Microprocessing and Microprogramming*, vol. 41, no. 5-6, pp. 409–423, 1995.
[66] A. Moga, T. Viero, B. Dobrin, and M. Gabbouj, "Implementation of a distributed watershed algorithm," in *Mathematical Morphology and Its Applications to Image and Signal Processing*, pp. 281–288, Kluwer Academic, Dordrecht, The Netherlands, 1994.
[67] D. Noguet, *Architectures parallèles pour la morphologie mathématique géodésique*, Ph.D. thesis, Institut National Polytechnique De Grenoble, Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs, Grenoble, France, Janvier 2002.
[68] A. Bieniek, *Divide-and-Conquer Parallelisation Methods for Digital Image Processing Algorithms*, vol. 10 of *VDI Fortschritt-Berichte*, VDI Verlag, Düsseldorf, Germany, 2000.
[69] M. Dubois, C. Scheurich, and F. Briggs, "Memory access buffering in multiprocessors," in *Proc. 13th Annual International Symposium on Computer Architecture (ISCA '86)*, pp. 434–442, Tokyo, Japan, June 1986.
[70] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in *Proc. 17th Annual International Symposium on Computer Architecture (ISCA '90)*, pp. 15–26, Seattle, Wash, USA, May 1990.
[71] F. Meyer and P. Maragos, "Multiscale morphological segmentations based on watershed, flooding, and eikonal PDE," in *Proc. 2nd International Conference on Scale-Space Theories in Computer Vision (Scale-Space '99)*, M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, Eds., vol. 1682 of *Lecture Notes in Computer Science*, pp. 351–362, Springer, Corfu, Greece, September 1999.
[72] P. Dokládal, R. Enficiaud, and E. Dejnožková, "Contour-based object tracking with gradient-based contour attraction field," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP '04)*, vol. 3, pp. 17–20, Montreal, Quebec, Canada, May 2004.
[73] F. Meyer and C. Vachier, "Image segmentation based on viscous flooding simulation," in *Proc. 6th International Symposium on Mathematical Morphology (ISMM '02)*, vol. 2, pp. 69–77, Sydney, Australia, April 2002.
[74] R. Haralick, "Digital step edges from zero crossing of second directional derivatives," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, no. 1, pp. 58–68, 1984.
[75] R. Kimmel and A. Bruckstein, "On edge detection integration and geometric active contours," in *Proc. 6th International Symposium on Mathematical Morphology (ISMM '02)*, vol. 2, Sydney, Australia, April 2002.
[76] C. Xu and J. L. Prince, "Snakes, shapes, and gradient vector flow," *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 359–369, 1998.
[77] K. Sobottka and I. Pitas, "Extraction of facial regions and features using color and shape information," in *Proc. 13th IEEE International Conference on Pattern Recognition (ICPR '96)*, vol. 3, pp. 421–425, Vienna, Austria, August 1996.
[78] A. Nayak and S. Chaudhuri, "Self-induced color correction for skin tracking under varying illumination," in *Proc. IEEE International Confererence on Image Processing (ICIP '03)*, vol. 3, pp. 1009–1012, Barcelona, Spain, September 2003.
[79] V. Veselý, "Fast Algorithms of Fourier and Hartley Transform and their Implementation in MATLAB," http://citeseer.ist.psu.edu/vesely98fast.html.
[80] H. Karner, M. Auer, and C. Ueberhuber, "Optimum complexity FFT algorithms for RISC processors," Tech. Rep. AURORA TR1998-03, Institute for Applied and Numerical Mathematics, Technical University of Vienna, Vienna, Austria, 1998.
[81] S. Osher and N. Paragios, Eds., *Geometric Level Set Methods in Imaging, Vision and Graphics*, chapter 2, Springer, New York, NY, USA, 2003.
[82] http://www.eece.unm.edu/xup/microblazeppc.htm.
[83] http://www.arm.com/miscPDFs/4491.pdf.
[84] http://support.intel.com/design/pentium4/datashts/24919805.pdf.

Eva Dejnožková is a research engineer with the Commissariat à l'Energie Atomique à Saclay, France, which she joined in 2004. She graduated with the highest degrees from the West Bohemia University, Pilsen, Czech Republic, in 1999, as an engineer specialized in industrial electronics. Afterwards, she specialized in hardware architecture for image processing, and obtained her Ph.D. degree from the School of Mines of Paris, France. She obtained a special distinction of the rector of the West Bohemia University. Her research interests include image processing and compression, and embedded and mobile computers architecture for image processing and compression.

Petr Dokládal is a research engineer at the Centre of Mathematical Morphology, the School of Mines in Paris, France. He graduated from the Technical University in Brno, Czech Republic, in 1994, as a telecommunication engineer and received his Ph.D. degree from the University of Marne la Vallée, France, in general computer sciences, specialized in image processing. His research interests include medical imaging, image segmentation, object tracking, and pattern recognition.

Computationally Efficient, One-Pass Algorithm for Morphological Filters

Petr Dokládal[a,*], Eva Dokládalová[b]

[a] *Center of Mathematical Morphology, Department Mathematics and Systems, Mines PARISTECH, 35, rue St. Honoré, 77 300 Fontainebleau Cedex, France*
[b] *Unité mixte de recherche CNRS-UMLV-ESIEE, UMR 8049, Université Paris-Est, Cité Descartes B.P.99, 93 162, Noisy le Grand Cedex, France*

Abstract

Many useful morphological filters are built as long concatenations of erosions and dilations: openings, closings, size distributions, sequential filters, etc. This paper proposes a new algorithm implementing morphological dilation and erosion of functions. It supports rectangular structuring element, runs in linear time w.r.t. the image size and constant time w.r.t. the structuring element size, and has minimal memory usage.

It has zero algorithm latency and processes data in stream. These properties are inherited by operators composed by concatenation, and allow their efficient implementation. We show how to compute in one pass an Alternate Sequential Filter (ASF^n) regardless the number of stages n.

This algorithm opens the way to such time-critical applications where the complexity and memory requirements of serial morphological operators represented a bottleneck limiting their usability.

Keywords: Mathematical morphology, Nonlinear filters, Alternate sequential filters, Real-time

1. Introduction

Since its introduction in late sixties, the Mathematical morphology provides a complete set of image processing tools from filtering [1, 2], multi scale image analysis [3] to pattern recognition [4, 5, 6]. They have been used in unrivalled number of applications [7, 8]. The most significant examples include biomedical and medical imaging, video surveillance, industrial control, video compression [9], stereology or remote sensing [10].

Nonetheless, not all useful operators can be easily implemented in real time with reasonable memory requirements. In demanding image-interpretation applications requiring a high correct-decision liability, one often uses robust but costly multi-criteria and/or multi-scale analysis.

These applications often consist of a serial concatenation of alternating atomic operators dilation and erosion with progressively increasing computing window called structuring element (SE).

Such operators cannot be parallelized due to the sequential data dependency of the individual atomic operators. The only possibility is to minimize the latency of each atomic operator and consider computing in stream. The latency minimization reduces the time to wait for individual pixel results. The stream computing allows transferring them immediately to the next atomic operator, as soon as they are available and before the entire image is processed. Thus, these atomic operators can work simultaneously, on data delayed in time. In such implementation, one has to sum the individual working memories of every atomic operator. Then also the memory may become penalizing for large, high-resolution images.

The work presented in this paper, aims to propose a new dilation/erosion algorithm with a constant processing time, low latency and low memory requirements for implementation of the individual atomic operators. Consequently, it allows to implement advantageously the following:

1. Alternate Sequential Filters (ASF) - that are a concatenation of openings and closings with a progressively increasing structuring element, useful for multi-scale analysis [1].

2. Size distributions (granulometries) - that are a concatenation of openings allowing to assess the size distribution of a population of objects [3, 11, 12].

3. Statistical learning - a selected set of morphological operators ξ_i can be separately applied to an image f. Then for every pixel $f(x, y)$, the vector of values $(\xi_i(f)(x, y))$ can serve as vector of descriptors for pixel-wise learning and classification [6]. Obtaining them may be computationally intensive.

1.1. Paper organization

The remainder of the Introduction lists the most known fast algorithms of morphological dilation and erosion and discusses their properties, followed by the explanation of Novelties in this paper.

The Preliminaries, Section 2, introduce the basic principles of dilations, erosions and their combinations.

The Section 3 outlines the Principle of the new Algorithm: i) the 2-D decomposition preserving sequential access to data and zero latency, ii) elimination of useless values, iii) the conversion of an anti-causal structuring element into a causal one, necessary to preserve the sequential access to data, and iv) the encoding used to reduce the memory requirements and acceleration of computations.

*Principal corresponding author

Sections 4 to 5 discuss the properties and section 6 presents the case of the four stage ASF as an example.

The paper concludes by Benchmarks, general Conclusions and Future extensions, sections 7 to 9. The commented pseudo code is given in the Appendix.

1.2. Existing work

The mathematical morphology relies on two fundamental, complementary operations: erosion and dilation. They are local, defined within a computing window, specified by the so-called structuring element (SE), characterized by its size, its shape and origin. It is well known that, with the increasing size of the SE, the direct implementation leads to an extremely high computing cost. Although the fastest existing algorithms [13, 14, 15, 16] concentrate mainly on the reduction of the number of comparisons, few of them deal with the latency and memory requirements [17]. Moreover, the minimization of comparison number is not always proportional to the overall performance improvement [15].

In the following paragraphs, we concentrate on the presentation of the existing state of the art. We discuss it from the classical point of view of complexity, based on the number of comparisons. We bring it face to face with the latency and the memory requirements. For each algorithm, we analyze the possibility of its stream implementation since it is a key feature allowing efficient chaining of the atomic operators.

Lets start by the definition of the used basic terms. Consider a system $Y = f(X)$ with X and Y the input and output data streams. By *latency* understand the distance between the same positions in the two streams. It is a dimensionless value, expressed in number of data samples. It is the sum of several factors:

1) *operator latency* - is induced by non causal operators due to the fact that the value to output depends on future signal samples. Consider a basic max filter y_j=max$(x_{j-w/2}, \ldots, x_{j+w/2})$. One cannot output y_j before having read all x_i until $x_{j+w/2}$,

2) *algorithm latency* - some algorithms continue reading the input even after all needed input data are available. Several morphological dilation/erosion algorithms run in two (forward and backward) data scans, e.g. [13, 18]. Typically, in [18], before processing one image line, one needs to read the entire line. For 2-D dilation by a rectangle, implemented separately in the horizontal and vertical direction, one would need to wait the bottom of the image before writing the result. In [13] these forward and backward scans can be done on w pixels long intervals.

For example, the naive implementation of the morphological dilation (Eq. 3) has a considerable computation complexity $\mathcal{O}(w\text{-}1)$ per pixel, with w the SE width (or area in 2-D), but no algorithm latency.

The operator latency - inherent to the operator - is incompressible. Consequently, the optimization effort should focus on the algorithm latency and the computational complexity.

The first concern related to the latency is therefore the time response of the system. Another concern related to the latency is the memory requirements. This can intuitively be explained by the fact that the latent (meaning "hidden") data need to be temporarily stored somewhere to not to get lost. Obviously, a large latency requires large storage. An interesting conclusion is that using larger SE will have larger memory requirements.

1.2.1. State of the art

The scientific community has adopted several approaches to speed up the erosion/dilation computation. The first one, we call *direct computation*, consists of a straightforward optimization of the computation given the SE shape.

The second approach relies on the *SE decomposition* into a sequence of reduced SE. Consequently, the optimization effort concentrates on the computation of this smaller SE. The special attention is paid to the SE decomposition into a series of *1-D* SE, very popular in numerous applications [19, 20]. It allows better data access, reuse of intermediate results and is easy to parallelize.

In the following, refer to Tab. 1 and 2 summarizing the properties of some algorithms cited below. By data memory understand the temporary storage for input or output data if the algorithm uses random data access. For instance the direct implementation needs random access to input data, whereas the output is written sequentially. It includes also the image transposition used by some algorithms. The working memory is any supplementary memory space required by the algorithm. It includes the data structures like FIFOs, LUTs, histograms, etc. Temporary constants, scalar variables, counters, etc, are omitted.

Direct 2-D computation. Optimized algorithms reduce the computing redundancy by using some well-suited data structures to keep the intermediate results. The most natural way is the approach used by Huang *et al.* [21] for median filtering, by Chaudhuri *et al.* [22] for rank-order filtering, and later by Van Droogenbroeck and Talbot [23]. They use a histogram to store the values within the span of the SE at some position in the image. During the translation of the SE over later image positions the histogram is updated by inclusion/deletion of the values of the entering/leaving points. The family of available shapes for the SE is arbitrary. On the other hand, using histogram makes that the input data have to be integers.

SE decomposition. It has soon become evident that the SE decomposition offers another possibility to obtain a fast implementation of more complex SEs both on specialized hardware as well as on sequential computers, and the literature soon became abundant see e.g. [24, 25, 26, 27, 28, 29, 30, 31]. The speedup is obtained by dividing the effort in two independent key aspects, an efficient decomposition and the algorithm used for computing the atomic operations.

Various types of decompositions have been proposed. Perhaps the most known decomposition of linear sets is the *linear decomposition* which comes from the associativity of the dilation, see Matheron [28]. Pecht [29] has proposed a more efficient logarithmic decomposition based on the *extreme set* of some SE. For example, for a polygon, the extreme set contains the vertices.

Table 1: 2-D algorithms comparison

Algorithm	SE type	Complexity per pixel	Algorithm Latency	Data Memory	Working Memory
Naive 2-D	User	$\mathcal{O}(WH)$	0	MN	0
Urbach-Wilkinson	User	$\mathcal{O}(N_c + \log_2(L_{max}(C)))$	MN	MN	$NH\log_2 W$
Van Droogen-broeck-Talbot	User	$\mathcal{O}(H \log_2(G))$ *	0	NH	WHG
This paper 2-D	Rect.	$\mathcal{O}(1)$	0	0	$2(NH+W)$

$W \times H$ = SE size (Width×Height); $N \times M$ = image size; G = number of gray levels; $L_{max}(C)$ = maximum chord length; N_c = number of chords; * square SE

Table 2: Fast 1-D algorithms comparison.

Algorithm	SE type	Comparisons per pixel	Algorithm Latency	Overall Memory
Naive 1-D	User	$W-1$	0	N
van Herk Gil-Wermann	Sym	$3 - \frac{4}{W}$	W	$N+2W$
Gil-Kimmel	Sym Even/Odd	$1,5 + \frac{\log_2 W}{W} + +\mathcal{O}(\frac{1}{W})$	W	$N+3W$
Lemire	Left	3	0	$N+W$
Lemonnier	Sym	nc	N	$2N$
Van Droogen-broeck-Buckley	Sym Even/Odd	nc	0	$2N+G$
This paper 1-D	User	$\mathcal{O}(1)$	0	$2W$

Sym = symmetric SE; Left = Left sided SE; User = user defined; W=SE size; N=line size; G = number of gray levels; nc = not communicated.

Van den Boomgaard and Wester [30] show that the Pecht decomposition can be improved for convex shapes. They propose a decomposition of an arbitrary shape into the union of convex shapes taken from a fixed collection of basis, efficiently decomposable shapes. Coltuc and Pitas [31] propose a factorization based algorithm running efficiently for 2^n signals. Soille *et al.* [27] propose an extension of the 1-D van Herk algorithm to 2-D. The SE is a line oriented in an arbitrary angle. The decomposition is obtained by saving the 2-D image as an 1-D array, and recomputing the pixel indices correspondingly to the given orientation of the line. Another efficient algorithm has been recently proposed by Urbach and Wilkinson [16]. It decomposes a flat, arbitrary-shape SE by using a set of 1-D chords. The min/max statistics of the chords are stored in LUT.

1-D algorithms. The 1-D algorithms compute the partial 1-D dilations after the SE decomposition into lines.

One of the earliest, and most often used 1-D algorithms, is the van Herk algorithm [13] proposed in 1992. The same algorithm completed by theoretical background was also published by Gil and Werman [32], and later improved by Gevorkian *et al.* [33] and Gil and Kimmel [14]. The computational complexity is independent of the SE size. It requires two passes on the input data: causal and anti-causal. Consequently, computing in stream is impossible. Another, similar algorithm was proposed in [34] using ring-type buffers. Recently, Clienti *et al.* [35] propose an interesting modification of the Van Herk algorithm reducing the memory to $2W$, implemented on an FPGA.

A different approach has been used by Lemonnier [18]. It identifies and propagates local extrema as long as it is required by the SE size. Again, two passes are needed: causal and anti-causal. Hence, the algorithm latency is N. The stream execution is impossible.

Van Droogenbroeck and Buckley [15] publish an anchor based algorithm for erosions and openings. The anchors are these portions of signal that remain unchanged by the operator. The algorithm gives good performance in terms of the computing time. The erosion can not run *in place* and stream processing is probably impossible. The principal disadvantage is in using histograms (suited only for integer values, and making the algorithm irregular).

Lemire proposes a fast, stream-processing algorithm for a left-sided SE [17], and later for symetric SE [36]. Both versions simultaneously compute 1-D dilation and erosion, run on floating point data and have low memory requirements and zero latency. However, even though the algorithms are supposed to run in stream, the intermediate storage of coordinates of local extrema actually represents a random access to the input data.

1.3. Latency issues

In order to asses the latency of 2-D algorithms we need to consider separately these different algorithm categories:

• For decompositions of rectangles using $R = H \oplus V$ (R=rectangle, H/V=horizontal/vertical segment respectively) the latency in 2-D is a multiplicative factor of the latency in 1-D and the image width. For two pass algorithms, e.g. Lemonnier [18], the 2-D latency equals one image frame. For locally two-pass algorithms, [13, 32, 33, 27] a specific decomposition would have to be found to optimize the latency in 2-D. The same holds also for other 1-D algorithms that in 1-D allow streaming processing [17, 36, 15].

• Regarding the direct computation in 2-D, though UW [16] could theoretically read/write the input/output images sequentially, the possibility of streaming processing is not mentioned; they also use random accesses to intermediate data. Van Droogenbroeck-Talbot [23] and the naive implementation write output sequentially, but use random accesses to input data.

• The computational complexity, used in the Tab. 1, may introduce to the result a supplementary delay - the *time to compute the result*. Whereas the two latencies are relative to the stream rate, the additional delay depends on the implementation and the computation platform. It can be i) negligeable as in most R=H⊕V decompositions, where the latency prevails, or 2) dominant like in the direct implementation with large SE or in 3-D.

1.4. Novelty of this paper

Although one can find several 1-D algorithms running with zero algorithm latency, none of the above cited algorithms combines all the features necessary for efficient implementation of composed operators in the form $\xi = \delta_{B_n}\varepsilon_{B_{n-1}} \ldots \delta_{B_2}\varepsilon_{B_1}$ for 2-D images.

Suppose the atomic operators δ, ε implemented using an algorithm with sequential access to data. This allows to run in parallel the entire ξ despite its internal sequential data dependence. If the atomic algorithm, in addition, has zero algorithm latency, then the entire chain ξ inherits the same properties: sequential data access and zero algorithm latency. This is an interesting property, since computing ξ suddenly becomes very efficient: in stream, with only the (further irreducible) operator latency of ξ. See the application example Fig. 4.

In this scope, the novelty of this paper is multiple. It is the only algorithm that combines all necessary features for efficient, parallel implementation of serial morphological operators. It uses a strictly *sequential* access to data, and can also run *in place*. The output is produced with *zero algorithm latency*. The algorithm runs in *linear time* w.r.t. the image size and *constant time* w.r.t. the SE size.

Its additional features include: very low *memory requirements*. A natural support of *floating point* data (not all previous algorithms can support floating point data). *The origin* can be arbitrarily placed within the structuring element, which is useful for even sized SE or specific SE decompositions.

2. Preliminaries

2.1. Morphological Dilation and Erosion

Let δ, ε: $\mathcal{L} \to \mathcal{L}$ be a dilation and an erosion, performed on functions $f \in \mathcal{L}$, defined as $f : D \to V$. Below assume $D = \text{supp}(f) = \mathbb{Z}^n$, $n = 1, 2, \ldots$ and $V = \mathbb{Z}$ or $\mathbb{R}$. δ_B, ε_B are parameterized by a structuring element B, assumed rectangular and flat i.e. $B \subset D$ and translation-invariant.

Functional (operating on functions) erosion and dilation by a flat SE defined by extension to functions of the Minkowski set addition/subtraction definitions are given by

$$[\delta_B(f)](x) = [\bigvee_{b \in B} f_b](x) \tag{1}$$

$$[\varepsilon_B(f)](x) = [\bigwedge_{b \in \widehat{B}} f_b](x) \tag{2}$$

where $\widehat{\ }$ denotes the transposition of the structuring element, equal to a set reflection $\widehat{B} = \{x \mid -x \in B\}$, and f_b denotes the translation of the function f by some vector $b \in D$. Hence, the definitions Eqs. (1, 2) can be implemented by

$$[\delta_B(f)](x) = \max_{b \in B} f(x - b) \tag{3}$$

$$[\varepsilon_B(f)](x) = \min_{b \in B} f(x + b) \tag{4}$$

Dilations and erosions combine to form other operators. We shall focus on combinations obtained by concatenations that this algorithm implements optimally.

The basic products obtained by concatenation[1] are opening $\gamma_B = \delta_B\varepsilon_B$ and closing $\varphi_B = \varepsilon_B\delta_B$. Hence from, one forms the so called Alternating Filters obtained as $\gamma\varphi$, $\varphi\gamma$, $\gamma\varphi\gamma$ and $\varphi\gamma\varphi$. The number of combinations obtained from two filters is rather limited. Other filters can be obtained by combining two *families* of filters. This leads to morphological Alternate Sequential Filters (ASF), originally proposed by [37], and studied in [1] Chap. 10. In general, it is a family of operators parameterized by some $\lambda \in \mathbb{Z}^+$, obtained by alternating concatenation of two families of increasing, resp. decreasing filters $\{\xi_i\}$ and $\{\psi_i\}$, such that $\psi_n \leq \ldots \leq \psi_1 \leq \xi_1 \leq \ldots \leq \xi_n$.

The most known ASF are those based on openings and closings, obtained by taking $\psi = \gamma$ and $\xi = \varphi$:

$$ASF^\lambda = \gamma^\lambda\varphi^\lambda \ldots \gamma^1\varphi^1 \tag{5}$$

starting with a closing, and

$$ASF^\lambda = \varphi^\lambda\gamma^\lambda \ldots \varphi^1\gamma^1 \tag{6}$$

starting with an opening.

This brief survey of theory allows to intuitively appreciate the complexity and the challenge involved by the usage of such long compound operators. If the algorithm does not deal at the same time with the memory management as well as with the latency, the overall performances could be (and generally they are) significantly lowered. We address this problem in Section 6 where we show how to efficiently implement the concatenation of dilations and erosions.

[1] to be read from right to left

3. Principle of the Algorithm

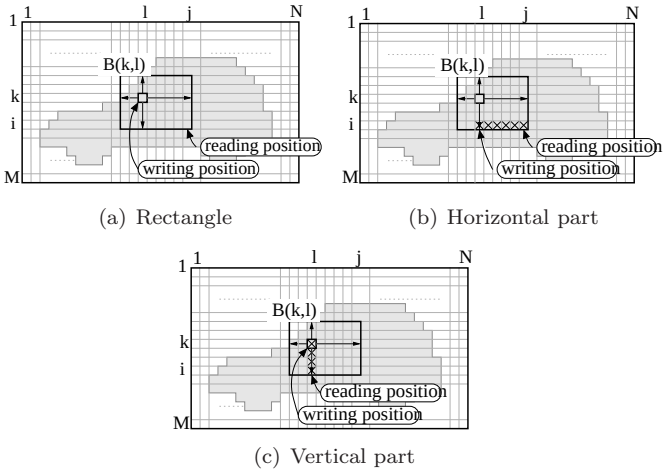
According to the algorithm classification presented in the Introduction, the proposed algorithm belongs to the *SE decomposition* approach using an improved *1-D dilation algorithm*.

3.1. Separation of 2-D into 1-D

Recall that separable operators are run in all directions separately. This requires intermediate data storage between individual runs. If the 1-D parts use sequential data access, it allows to compose n-D dilations also using sequential data access. This eliminates the necessity of intermediate data storage.

The input image is read in the raster scan order, line by line. Every line is dilated horizontally. The result of the horizontal dilation is immediately read, pixel by pixel, by the vertical dilation in the corresponding column. The result of the vertical dilation part is written to the output. The output image is also written in the raster scan order.

Fig. 1a illustrates the computation of the result at position (k, l). Assume that the input data have already been read until line i, column j. The ongoing computations (depicted by $\times$) are: the horizontal dilation part (Fig. 1b) is running on line i, with the reading position (i, j) in the input image, writing position (i, l), immediately read by the vertical dilation (Fig. 1c) with reading position (i, l) and writing position (k, l), directly written to the output. The lines 1 to i–1 have already been horizontally dilated, and all columns have already been vertically dilated up to the line k.



(a) Rectangle (b) Horizontal part

(c) Vertical part

Figure 1: Separation of computing: (a) rectangular element into (b) the horizontal and (c) vertical part. $\times\times$ denote the data stored in the queue of the corresponding line or column.

3.2. 1-D algorithm

3.2.1. Elimination of useless values

An efficient coding of the function profile can avoid a number of comparisons during the computation of a dilation or an erosion. One can drop all values that will never take over in the result of the max or min, Eqs. (3, 4).

Consider a 1-D, connected structuring element B containing its origin. Then, computing $\delta_B f(x)$ needs only those values of $f(x_i)$ that can be seen from x when looking over the topographic profile of f. The valleys shadowed by mountains contain unneeded values, see Fig. 2. Notice that the masked values depend on f, and not on B.

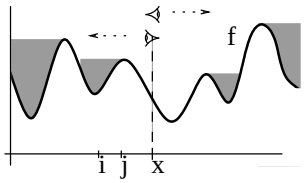


Figure 2: Computing the dilation $\delta_B f(x)$: Values in valleys shadowed by mountains when looking from x over the topographic relief of f are useless.

Now, lets place ourselves in the context of streaming algorithms. For simplicity assume a *causal* SE, i.e. containing its origin at the right hand side. For causal SE, one only needs to look leftwards, over the past samples. The search of the useless values can be formalized by the Prop. 1 showing that values useless at some time instant x remain useless also for the "future".

Proposition 1. [*Useless values*] *In computing the dilation $\delta_B f$, with $f : \mathbb{Z}^+ \to \mathbb{R}$, by some causal, connected structuring element B (a linear segment) containing its origin, no $f(i)$ such that $f(i) \leq f(j)$, and $i < j$, will influence the dilation*

$$\delta_B f(x), \text{ for } \forall x \geq j \tag{7}$$

Proof. *From Eq. 3, any x such that $i < j \leq x$, if $i \in B(x)$ then $j \in B(x)$. If $f(i) < f(j)$, then $f(i) < \max_{b \in B} f(x - b)$, and $f(i)$ has no impact on the dilation result.*

This means that all $f(i)$ such that

$$f(i) \leq f(j), \text{ with } i < j, \tag{8}$$

may be dropped from the computations.

This is a strong proposition that allows a considerable reduction of the computational redundancy. One comparison $f(i) \leq f(j)$, done upon reading $f(j)$, avoids computing $j-i$ useless comparisons for any later $B(x)$ that covers i and j.

Two important points are to be noticed.

1. $\forall x \geq j$ in Eq. 7 means that all values that become useless at the position j remain useless in the future, $\forall x \geq j$.

2. Using a bounded and causal $B \subset \mathbb{Z}$, i.e. an interval $B(x) = [x - b, x]$, with $b < \infty$, means that for computing $\delta_B(x)$, one can also discard all values outside the SE span, i.e. $f(x_i)$, with $x_i < x - b$.

Remark. This proposition does not hold for non causal SE. The values useless at time x may become useful for some $k > x$. This algorithm utilizes the commutation of dilation with translation to convert an anti-causal SE to a causal one.

3.2.2. Anti-causal to causal SE conversion

Every SE B, $B \subset D$ is equipped by an origin $x \in D$. Assuming a sequential access to the input data, the dilation $\delta_B f(x)$ depends of points read before but also after x. We say that B is non causal.

One can transform a non causal SE to a causal SE by utilizing the property that dilation commutes with translation ($t \in D$)

$$\delta_{B+t} f(x) = \delta_B f(x - t) \tag{9}$$

The translation consists of writing the result at the correct place in the output. The horizontal and the vertical shifts are handled by the 1-D horizontal or vertical dilation part, implemented by the Fnct. 1 page 9.

3.2.3. Function coding

Similarly as binary objects can be coded by using the distance to their boundaries, functions need to be coded by computing the distance to every change of the value. Using the Prop. 1 and relative indexing, the samples $f(x)$ used in computation $\delta_B f(x_i)$ are coded by pairs (*distance, value*) as given in Fig. 3.

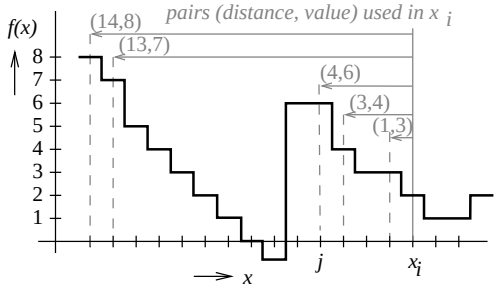


Figure 3: Function coding. The useless values are discarded.

The arrows indicate those values that enter in the computation of $\delta_B f(x_i)$. The values non indicated by an arrow are smaller or equal to $f(j) = 6$ and have no impact on the result $\delta_B f(x)$, for $x \geq j$.

The Eq. 8 is used by the 1-D dilation algorithm to exclude from the computation all useless values.

4. Algorithm Complexity and Latency

In this section we shall analyze the latency and the complexity of the algorithm.

4.1. Latency

The overall algorithm latency is function of two factors: i) the latency of the 1-D dilation (Fnct. 1), and ii) the latency of the 2-D decomposition (Algorithm 2).

• 1-D dilation : The Function 1 writes the output as soon as the reading position *rp* reaches the last position covered by the structuring element (code lines 6 to 7). This corresponds to the last output-to-input data dependency position, i.e. the operator latency.

Remark: The *while* loop, clearing from the FIFO the useless values, operates on past signal samples. Consequently, it doesn't enter in the latency count. The latency of the Function 1 is therefore strictly equal to the operator latency.

• 2-D dilation : The 2-D dilation is decomposed in the way that result of the horizontal 1-D dilation is directly fed to the corresponding 1-D vertical dilations. Their results are recombined into the output stream. Therefore, the algorithm latency of the 2-D decomposition is zero.

4.2. Computation complexity

The 2D_Dilation algorithm iterates over all coordinates of the output image. The inner complexity of the 2D algorithm is $\mathcal{O}(N)$, where N is the number of pixels in the image. At every coordinate, the 2D_Dilation calls twice the 1D_Dilation function: once for the vertical dilation and once for the horizontal dilation part.

The 1D_Dilation part (Fnct. 1) contains a sequence of $\mathcal{O}(1)$ operations, and one *while* loop (lines 1-2), clearing useless values from the FIFO. We shall see that this *while* loop is executed at most once per pixel, making the complexity of the 1D_Dilation algorithm constant per pixel.

Every incoming pixel is stored in the FIFO once and only once (line 5). Every pixel is cleared from the FIFO once and only once, either i) when it becomes "too old", i.e. uncovered by the current SE span (lines 3-4) or ii) when it gets masked by another, higher value (lines 1-2).

The deletion (lines 1-2) of every pixel can be delayed. Delete pixels from FIFO later or sooner has no impact on the algorithm complexity. However, it occurs that several pixels are deleted at the same time. This implies using the loop *while* (lines 1-2). The loop iterates at most once per pixel. Other pixels that become "too old" are deleted at lines 3-4. Hence, both ways of deletion have the same complexity $\mathcal{O}$ (1) per pixel.

This allows to make the following conclusions:

1) The FIFO size is upper-bounded by the SE width. This determines the memory requirements (detailed in the next section).

2) For every pixel, the number of the iterations of the *while* loop is lower-bounded by zero and upper-bounded by the SE width.

3) The average number of iterations of the *while* remains in $[0, 1]$ per pixel.

4) The worst-case complexity of the 1D_Dilation per pixel is bounded by $\mathcal{O}(W)$.

Hence, we shall conclude that the overall complexity of the 2-D dilation algorithm is $\mathcal{O}(N)$, i.e. linear w.r.t the size of the image (N pixels) and constant w.r.t the SE size.

Note: Although both ways of deletion have the same complexity $\mathcal{O}(1)$, they do not have the same cost (in terms of instruction count). The deletion by shadowing is slower in C because of the overhead of the loop *while*. The different timings obtained on various data (constant, random or natural images) are due to this overhead. For nonincreasing intervals (e.g. a constant image - see Benchmarks) the loop (lines 3-4) never executes. The probability of either deletion being data dependent explains the slight variation of the execution time on the image content.

5. Memory Requirements

In 1-D, the FIFO size is upper-bounded by the width of the SE, which is the memory-worst case encountered

whereever there are no useless data to eliminate. This occurs at all monotonically decreasing intervals of the signal that are longer than the SE width. This is equivalent to results obtained in [15], where for computing $\varepsilon_B f(x)$, only the values $f(x_i)$, with $x_i = B(x)$, are needed. Similar results hold for the dilation.

In 2-D, the memory requirements are given by the decomposition of the rectangle as $R = H \oplus V$ (R=rectangle, H/V=horizontal/vertical segment respectively). The raster-scan data access makes that the computations window (the SE) slides over the image from left to right and from top downwards.

The vertical part of the 2-D dilation runs at all columns simultaneously, one pixel per each column at a time. All columns have the memory-worst case equivalent to the height of the SE.

Consider an erosion (or a dilation) of an $N \times M$ image (width by height) by a $W \times H$ rectangular (width by height) SE. The memory requirements M are

$$M = 2(NH + W)$$

This means, N memory blocks of size $2H$ (vertical part) and one memory block of size $2W$ (horizontal part). The multiplicative factor 2 comes from the fact that the stored data are indexed by their coordinates (see Fig. 3, and Fnct. 1, line 5).

For example, an erosion of an 800×600 image by a 20×20 square will require 2×(800×20+20)=32,040 bytes. Compared to this, storing an 800×600 image is costly, requiring 480,000 bytes (with 1 byte/pixel coding). Neither the input nor the output image need to be stored in memory.

The memory requirements graphically correspond to storing the image data from the lines currently intersected by the SE.

6. Application

In the following we give as example the implementation of an ASF^λ given by Eq. (5). Rewrite the filter as a concatenation of erosions and dilations

$$ASF^\lambda = \delta_{B_\lambda}\varepsilon_{B_\lambda}\varepsilon_{B_\lambda}\delta_{B_\lambda} \ldots \delta_{B_1}\varepsilon_{B_1}\varepsilon_{B_1}\delta_{B_1} \qquad (10)$$

and reduce it into its canonical form

$$ASF^\lambda = \delta_{B_\lambda}\varepsilon_{B_\lambda \oplus B_\lambda}\delta_{B_\lambda} \ldots \delta_{B_1}\varepsilon_{B_1 \oplus B_1}\delta_{B_1}$$

This ASF^λ can be implemented in a stream in one raster scan of the input image. The writing position of the preceding operator in the cascade becomes the reading position of the following operator. The operator latency of the entire ASF will be given by the one introduce by the result of Minkowski sum of all SE in Eq. 10, that is $\bigoplus_{i=1}^{n} B_i$.

Figure 4 illustrates the propagation of real image data through an ASF^4 after having read approximately one third of the input image. The SE is a square of size s+1 for the s-th stage. The individual operators, with sequential data dependence, are running simultaneously. There is no intermediate data storage between the stages; the intermediate results are pipelined.

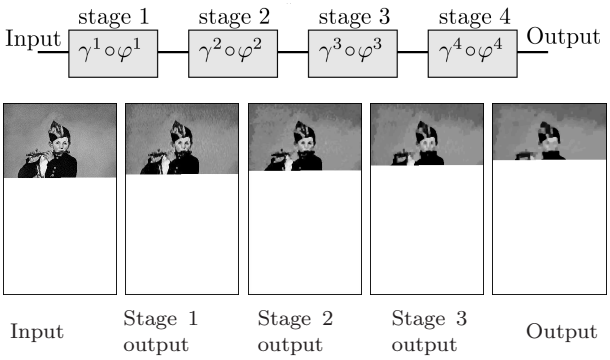


Figure 4: Propagation of data throughout ASF^4 after having read approximately one third of input image (Manet's painting "Le fifre")

Table 3: Execution time in ms for 2-D dilation of the 'mountain.pgm' image (800×600) by a 21×21 square for various data coding types.

Data type / CPU type	int	float	unsigned char	double
Intel Core2 Duo 2.4GHz (32 bits)	17.49	18.05	25.77	20.32
Dual Core AMD Opteron 2.4GHz (64 bits)	22.34	23.64	25.02	22.06

7. Benchmarks

This section illustrates the execution time of this algorithm, w.r.t. various criteria, measured on an Intel Core 2 2GHz CPU, with 2GB 800MHz Dual Port RAM, running Linux. The time reported below is the processor time spent in the dilation/erosion algorithm as reported by a profiler (obtained as a mean after several runs to reduce inaccuracy).

The first experiment, see Fig. 5, illustrates the running time w.r.t. the content of the image. We have used a constant and a white-noise image to illustrate the fastest and the worst-case running time, and a natural image to illustrate the "expected" running time on a natural scene. The measured time follows a linear function of the image size. Note also that the performance on the natural image actually coincides with the worst case performance obtained on the white noise. (The various sizes of the natural image were obtained by tiling side to side the original photography mountain.pgm from [38].)

We have evaluated the performance of the algorithm against different data types, see Tab. 3. The best performance has been obtained for the word width corresponding to the used CPU architecture, i.e. the integer and float for a 32 bit CPU, and the double for a 64 bit CPU. The penalty obtained for the unsigned char (8 bits) is due to inefficient memory access on both architectures.

We have compared our algorithm with Urbach-Wilkinson[2] [16] and Van Droogenbroeck-Buckley[3] [15], see Fig. 6.

The best timing was obtained with [15], the worst with [16], which allows - on the other hand - SE of ar-

[2]code courtesy of Erik Urbach
[3]code available at [38]

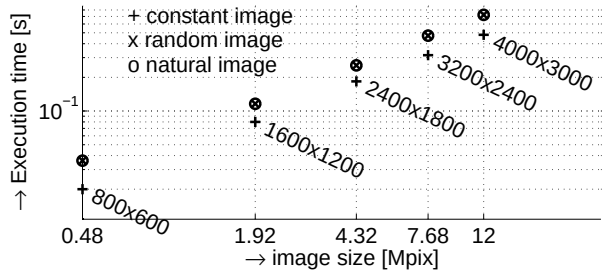


Figure 5: Execution time of erosion, with respect to the content of the image. Data read from memory.

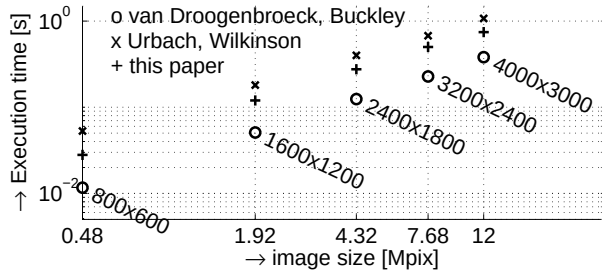


Figure 6: Execution time of erosion, with respect to the size of image. Structuring element 21×21. Data read from memory.

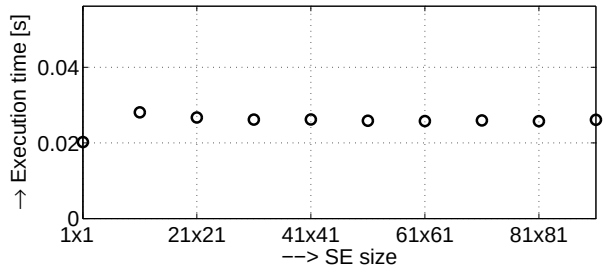


Figure 7: Execution time of erosion with respect to the size of structuring element. Image size 800×600.

bitrary shapes. The algorithm from this paper remains competitive with the speed of the other algorithms, even though the speed has been traded off for memory consumption and latency.

Finally, the last experiment, see Fig. 7, illustrates that the complexity of the algorithm is independent of the SE size. The SE is a centered, $10k{+}1 \times 10k{+}1$ square, for $k = 0 \ldots 9$.

Note: The FIFO queues are implemented in C by using pointer-addressed arrays.

8. Conclusions

This paper proposes a new algorithm for functional dilation or erosion by a flat, rectangular structuring element for 2-D data (easily extensible to n-D images, and SE in form of n-D boxes).

The algorithm has zero algorithm latency and strictly sequential access to data. The combination of these two properties allows their inheritance to operators composed by concatenation. The entire concatenation chain, despite its internal sequential data dependence, can run simultaneously. The algorithm runs in linear time w.r.t. the image size and constant time w.r.t. the SE size.

Regarding serial filters, if all the operators in the concatenation run simultaneously - then result can also be obtained in constant time w.r.t. the length of the concatenation.

The algorithm has low memory requirements, which eases its implementation on systems with space constraints, such as embedded or mobile devices, intelligent cameras, etc. The linear time and zero latency allow efficient implementations on demanding industrial systems with severe time constraints.

Even though the speed is not the principal concern here, the algorithm remains competitive in term of execution time compared to the recent proposed fast implementations.

9. Future extensions

9.1. Other SE shapes

The algorithm, described above with rectangles, can also be extended to other shapes decomposable into linear segments (e.g. polygons as in [27]).

9.2. Spatially variant SE

This paper is the third step of a wider work towards an efficient implementation of morphological operations with spatially variant structuring elements that are useful for adaptive filters. The first step has been the stream implementation of dilation/erosion of sets [39]. It has the same algorithmic properties: zero latency and optimal memory, sequential access to data. The second step was the extension to the functional morphology; preliminary results have already been published: 1-D spatially variant morphology [40] and approximations of 2-D spatially variant rectangles [41].

The present algorithm is a simplified version of [41] limited from spatially variant to translation invariant SE. This simplification has brought a 10× speed increase.

The goal is to obtain an algorithm for spatially-variant functional dilations and erosions with structuring elements of unconstrained shapes.

9.3. Real-time HW accelerator

This algorithm can be easily implemented as a finite state machine, interesting for HW implementation. The sequential access allows to read the image from a camera, process it, and write it out for further processing or visualization. This allows processing large, or *infinite* industrial images without storing them in the memory.

Acknowledgements

The authors wish to thank the anonymous reviewers for their comments and remarks that allowed to improve the paper.

Appendix

Notation convention and preliminaries

$\leftarrow$ shall denote the assignment and $=, <, \leq, ...$ tests. Curly braces denote a collection of values, e.g. $A \leftarrow \{5, 8\}$. To obtain an element in a collection by its index we use brackets, e.g. $A[1] = 5$. The empty set is denoted by $\{\}$. In a function call, parentheses denote the collection of arguments; funct() is a function call without arguments.

FIFO: The algorithm uses FIFO (First In First Out) queues. The FIFO supports the following operations: *push* - insert a new element, *pop* - retrieve the oldest element, and *dequeue* - retrieve the most recent element, and queries: *front* read the oldest element, *back* - read the newest element. The operations modify the content of the FIFO whereas the queries do not. `fifo`$\leftarrow\{\}$ initializes the fifo to empty.

In this algorithms, the elements inserted/read into/from the fifo are always couples {*value*, *position*}. Hence, e.g. the query `fifo.front()[1]` yields the *value* of the oldest element in the queue.

Input/output images are assumed 2D, read and written in the raster scan order one pixel at a time by x $\leftarrow$ in_stream.read() and out_stream.write(x). The reading/writing position is always implicitly incremented by 1.

Algorithm Description

This section details the algorithm principles in link to the algorithm pseudo-code, see page 9.

9.4. 2-D Dilation Algorithm

The 2-D Dilation, Algorithm 1, is to decompose the 2-D SE into columns and to assemble the partial 1-D computations into a 2-D stream, cf. Fig. 1.

The horizontal and vertical dilation parts are computed by the same function 1D_DILATION. It encodes the input data from the current line or column and stores them in the FIFO. There is one FIFO for the horizontal part dilation - `h_fifo`. For the vertical part, there is an $1 \ldots N$ array - `v_fifo` - one FIFO per image column.

The input image is read at lines 10 to 12. Missing data (to the right of the image) are completed by the padding constant, line 14. The horizontal dilation is computed at line 16.

If the horizontal dilation part outputs a valid (non empty) value, line 19, it is sent to the vertical dilation part, computed by the same function, line 20. The vertical dilations also may require padding - typically below the image - where the horizontal dilation is not called. Instead, dFx is directly set to the padding value, line 18.

Provided the vertical dilation outputs a valid result, line 21, it is directly written to the output image, line 22.

9.5. 1-D Dilation Function

Assume computing $dF{=}\delta_B F$, where F, $dF : [1,\ldots,N] \rightarrow \mathbb{R}$. The structuring element B is a linear segment, `SE1 + SE2 + 1` pixels long, with `SE1`, `SE2` the offsets of the origin from the left- or the right-most end.

Calling conventions: The function `1D_Dilation`, see Function 1, page 9, is a function computing one sample of `dF`, to be written at writing position `wp`.

Upon every call, `rp` must be incremented by one by the calling function. Similarly, every time that `1D_Dilation` outputs a valid sample, `wp` is to be incremented by one.

The current fifo queue needs to be passed by reference.

Principle: The function proceeds in three steps:

1. *Dequeue all smaller or equal values*, lines 1 to 2. Removes from the FIFO all values that become useless.
2. *Delete too old value*, lines 3 to 4, removes from the FIFO the value that gets uncovered by the current SE $B(wp)$.
3. *Enqueue the current sample* in the FIFO in the form of a couple {*value, position*}.
4. Provided enough data have been read, line 6, return the dilation result `dF`, line 7. At any moment, this value is found at the oldest position of the FIFO.

Note: To obtain erosion instead of dilation:
1) Fnct. 1, line 1, replace $\leq$ by $\geq$.
2) Alg. 2, line 1, set the padding constant PAD to ∞.

Algorithm pseudo-code

Function 1: `dF` $\leftarrow$ `1D_DILATION`(*rp, wp, F, SE1, SE2, N, fifo*)

Input: rp, wp - reading/writing position; F - input signal value (read at rp); SE1, SE2 - SE size towards left and right; N - length of the signal; fifo - the FIFO

Result: dF - output signal value (to be written at wp)

```
   // Dequeue all queued smaller or equal values
1  while fifo.back()[1] ≤ F do
2  |  fifo.dequeue()

   // Delete too old value
3  if (wp - fifo.front()[2] > SE1) then
4  |  fifo.pop()

   // Enqueue the current sample
5  fifo.push({F, rp})

6  if rp = min (N, wp + SE2) then
7  |  return ( fifo.front()[1] ) ;      // return a valid value
8  else
9  |  return ({}) ;                    // return empty
```

References

[1] J. Serra. *Image Analysis and Mathematical Morphology*, volume 2. Academic Press, NY, 1988.

[2] E. Dougherty. *Mathematical morphology in image processing*. Taylor and Francis, Inc., 1992.

[3] P. Maragos. Pattern spectrum and multiscale shape representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):701–716, 1989.

[4] J. Serra. Morphological filtering: an overview. *Signal Processing*, 38(1):3–11, 1994.

[5] S. Mukhopadhyay and B. Chanda. An edge preserving noise smoothing technique using multiscale morphology. *Signal Processing*, 82(4):527–544, 2002.

[6] A. Cord, D. Jeulin, and F. Bach. Segmentation of random textures by morphological and linear operators. In *Proc. International Symposium on Mathematical Morphology ISMM*, pages 387–398, 2007.

[7] R. Haralick, S. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(4):532–550, 1987.

[8] M. Wilkinson and J. Roerdink, editors. *Mathematical Morphology and Its Application to Signal and Image Processing*. Proc. 9th International Symposium on Mathematical Morphology. Springer, 2009.

[9] P. Salembier, P. Brigger, J. Casas Montse Pardas, and J. Casas. Morphological operators for image and video compression. *IEEE Trans. on Image Processing*, 5:881–897, 1996.

[10] P. Soille and M. Pesaresi. Advances in mathematical morphology applied to geoscience and remote sensing. *IEEE Trans. on Geoscience and Remote Sensing*, 40(9):2042 – 2055, 2002.

[11] R. Sabourin, G. Genest, and F. Prêteux. Off-line signature verification by local granulometric size distributions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(9):976–988, 1997.

[12] L. Vincent. Granulometries and opening trees. *Fundamenta Informaticae*, 41(1-2):57–90, January 2000.

[13] M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recog. Letters*, 13(7):517–521, 1992.

[14] J. Gil and R. Kimmel. Efficient dilation, erosion, opening, and closing algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1606–1617, 2002.

[15] M. Van Droogenbroeck and M. Buckley. Morphological erosions and openings: Fast algorithms based on anchors. *J. Math. Imaging Vis.*, 22(2-3):121–142, 2005.

[16] E. Urbach and M. Wilkinson. Efficient 2-D Grayscale Morphological Transformations With Arbitrary Flat Structuring Elements. *IEEE Trans. Image Processing*, 17(1):1–8, 2008.

[17] D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *Nordic J. of Computing*, 13(4):328–339, 2006.

[18] F. Lemonnier and J.-C. Klein. Fast dilation by large 1D structuring elements. In *IEEE International Workshop on Nonlinear Signal and Image Processing, Halkidiki, Greece*, 1995.

[19] L. Najman. Skew detection. European Patent, 2002. Filled at 27, 2002 as a European filing the French Patent Office.

[20] B. Obara. Identification of transcrystalline microcracks observed in microscope images of a dolomite structure using image analysis methods based on linear structuring element processing. *Comput. Geosci.*, 33(2):151–158, 2007.

[21] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Trans. Acoustics, Speech and Signal Processing*, 27(1):13–18, February 1979.

[22] B. Chaudhuri. An efficient algorithm for running window pel gray level ranking 2-D images. *Pattern Recogn. Letters*, 11(2):77–80, 1990.

[23] M. Van Droogenbroeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recog. Letters*, 17(14):1451–1460, 1996.

[24] X. Zhuang. Decomposition of morphological structuring elements. *J. of Math. Imaging and Vis.*, 4:5–18, 1994.

[25] X. Zhuang and R. Haralick. Morphological structuring element decomposition. *Computer Vision, Graphics, Image Processing*, 35:370–382, 1986.

[26] G. Anelli, A. Broggi, and G. Destri. Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(2):217–224, 1998.

[27] P. Soille, E. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(5):562–567, 1996.

[28] G. Matheron. *Random Sets and Integral Geometry*. John Wiley and Sons, New York, 1975.

[29] J. Pecht. Speeding up successive minkowski operations. *Pattern Recog. Letters*, 3(2):113–117, 1985.

[30] R. van den Boomgaard and D. Wester. Logarithmic shape decomposition. In C. Arcelli, L. P. Cordella, and G. Sanniti di Baja, editors, *Aspects of Visual Form Processing*, pages 552–561. World Scientific Publishing Co., Singapore, 1994. Capri, Italy.

[31] D. Coltuc and I. Pitas. On fast running max-min filtering. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(8):660–663, 1997.

[32] J. Gil and M. Werman. Computing 2-D Min, Median, and Max Filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):504–507, 1993.

[33] D. Gevorkian, J. Astola, and S. Atourian. Improving Gil-Werman Algorithm for Running Min and Max Filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(5):526–529, 1997.

[34] T. Miyatake, M. Ejiri, and H. Matsushima. A fast algorithm for maximum-minimum image filtering. *Systems and Computers in Japan*, 27(13):74–85, 1996.

[35] C. Clienti, M. Bilodeau, and S. Beucher. An efficient hardware arhcitecture without line memories for morphological image processing. In *Advanced Concepts for Intelligent Vision Systems*, 2008.

[36] D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lowerbound. *Pattern Recognition*, 42:2169–2180, 2009.

[37] S. Sternberg. Grayscale morphology. *Comput. Vision Graph. Image Process.*, 35(3):333–355, 1986.

[38] R. Dardenne and M. Van Droogenbroeck. The libmorpho library. Available for download from http://www2.ulg.ac.be/telecom/research/libmorpho.html.

[39] H. Hedberg, P. Dokládal, and V. Öwall. Binary Morphology with Locally Adaptive Structuring Elements: Algorithm and Architecture. *IEEE Transactions on Image Processing*, 18(3), 2009.

[40] P. Dokládal and E. Dokládalova. Grey-Scale 1-D dilations with Spatially-Variant Structuring Elements in Linear Time. In *European Signal Processing Conference*, 2008.

[41] P. Dokládal and E. Dokládalova. Grey-scale Morphology with Spatially-Variant Rectangles in Linear Time. In *Advanced Concepts for Intelligent Vision Systems*, 2008.

Algorithm 2: 2D_Dilation

Input: in_stream - input image pixels stream
M,N - height, width of the image
SE1, SE2, SE3, SE4 - struct. element size
Result: out_stream

```
1  const. PAD ← 0 ;      // Set the Padding Constant
2  vfifo ← array [1..N] of FIFO ; // array of N empty
   FIFOs for the vertical dilation part
3  line_rd ← 1 ;                  // read line counter
4  line_wr ← 1 ;                 // written line counter
   // iterate over all image lines
5  while line_wr ≤ M do
6      hfifo ← FIFO ; // FIFO for the horizontal part
7      col_rd ← 0 ;              // read column counter
8      col_wr ← 1 ;          // written column counter
       // iterate over all columns
9      while col_wr ≤ N do
           // horizontal dilation on the line_wr line
10         if line_rd ≤ M then
11             if col_rd < N then
12                 F ← in_stream.read()
13             else
14                 F ← PAD ;      // Padding constant
15             col_rd ← min(col_rd +1, N)
16             dFx ← 1D_Dilation (col_rd, col_wr, F,
               SE1, SE3, N, hfifo)
17         else
18             dFx ← PAD ;        // Padding constant
           // vertical dilation of the col_wr column
19         if dFx ≠ {} then
20             dFy ← 1D_Dilation (min(line_rd,M),
               line_wr, dFx, SE2, SE4, M, vfifo[col_wr
               ])
21             if dFy ≠ {} then
22                 out_stream.write(dFy)
23             col_wr ← col_wr +1
24     line_rd ← line_rd+1
25     if dFy ≠ {} then
26         line_wr ← line_wr +1
```

Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Jan Bartovský · Petr Dokládal · Eva Dokládalová · Michel Bilodeau · Mohamed Akil

Real-Time Implementation of Morphological Filters with Polygonal Structuring Elements

Received: date / Revised: date

Abstract In mathematical morphology, circular structuring elements (SE) are used whenever one needs angular isotropy. The circles – difficult to implement efficiently – are often approximated by convex, symmetric polygons that decompose under the Minkowski addition to 1-D inclined segments.

In this paper, we show how to perform this decomposition efficiently, in stream with almost optimal latency to compute gray-scale erosion and dilation by flat regular polygons. We further increase its performance by introducing a spatial parallelism while maintaining sequential access to data.

We implement these principles in a dedicated hardware block. Several of these blocks can be concatenated to efficiently compute sequential filters, or granulometries in one scan. With a configurable image size and programmable SE size, this architecture is usable in high-end, real-time industrial applications. We show on an example that it conforms to real-time requirements of the 100Hz 1080p FullHD TV standard, even for serial morphological filters using large hexagons or octagons.

Keywords Mathematical Morphology, Hardware Implementation, Alternating Sequential Filter, Parallel Computation, Polygonal Structuring Element

J. Bartovský
Faculty of Electrical Engineering, University of West Bohemia, Pilsen, Czech Republic.
Computer Science Laboratory Gaspard Monge, ESIEE Paris, University Paris-Est, Noisy-le-Grand, France.
E-mail: j.bartovsky@esiee.fr

E. Dokládalová, and M. Akil
Computer Science Laboratory Gaspard Monge, ESIEE Paris, University Paris-Est, Noisy-le-Grand, France.
E-mail: {e.dokladalova, m.akil}@esiee.fr

P. Dokládal and M. Bilodeau
Centre for Mathematical Morphology, Mines ParisTech, Fontainebleau, France.
E-mail: {petr.dokladal, michel.bilodeau}@mines-paritech.fr

1 Introduction

Since its first introduction in the late 1960's, mathematical morphology has becom in the field of image processing a useful tool for analysis of the shape or the form of spatial structures [17, 23, 24]. Over time it has found its application as a widely-used image processing technique [9, 18].

Thanks to the recent technological development of sensors, the resolution of images increased to tens of megapixels. Certain morphological operations, e.g., top-hat transformation, ultimate openings, granulometry, alternating sequential filters (ASF) [25], etc., on such large images require a large structuring element (SE), since its size should be proportional to the size of the image and its contents.

Existing hardware implementations either support rectangles using SE decomposition (fast computation, but angular anisotropic), or support arbitrarily-shaped SEs at the cost of significant performance decrease. Our work supports polygonal SEs at the performance rate of the rectangular SEs.

The paper is organized as follows: Section 2 makes a short survey of existing morphological algorithms and architectures. Section 3 outlines the basic aspects of morphological dilation and erosion, and show how to decompose the polygons into a set of lines. Section 4 describes the algorithm, and its use to decompose polygons while preserving the sequential access to data, minimal memory consumption and latency. Section 5 gives the functional implementation of the algorithm. The principal result, a parallel version using two levels of parallelism (temporal and spatial) is presented in Section 6. Finally, Section 7 presents experimental results obtained on an FPGA.

2 State of the art

First algorithms of morphological dilation were based on the definition (see Sec. 3), efficient for small structuring elements. The high computational complexity of large or arbitrarily-shaped SE motivated the search of decompositions, either by i) separation into lower dimensions, or ii) by

decomposition into atomic shapes, using the Minkowski addition. This section reviews the literature on the most known algorithmic decompositions and most efficient implementations.

i) The separation into lower dimensions allows the decomposition of n-dimensional rectangular SE into 1-D segments. The simplest method to compute 1-D dilation is an exhaustive search for maximum in the scope of SE B according to the definition (Eq. 1). Clearly, this naive solution tends to need a large number of comparisons. The number of comparisons is considered as a metric of algorithm complexity, so the naive algorithm has complexity $\mathcal{O}(l)$ as it has to carry out $l-1$ comparisons for l pixel long SE. Such complexity suggests that naive algorithm is inefficient for any large SEs. Pecht [20] proposed a method to decrease complexity based on logarithmic SE decomposition, thereby achieving $\mathcal{O}(\lceil \log_2(l) \rceil)$ complexity.

The first 1-D algorithm that reduced complexity to the constant is often referred to as HGW (it was published simultaneously in two papers: van Herk [28], and Gil and Werman [12]). The computation complexity is constant, i.e., of $\mathcal{O}(1)$, which means the upper bound of computation time is independent of the SE size. The HGW algorithm uses two buffers, which are filled by the forward, and backward, propagation of local maxima, respectively. Both buffers are then merged into the result. The major drawback of this algorithm is the requirement of two data scans: forward and reverse (so-called causal and anti-causal), which need temporary input data storing.

Lemonnier and Klein [16] propose another $\mathcal{O}(1)$ algorithm that also identifies local extrema and propagates their values. Again, the limiting forward and reverse scans are needed for every non-causal SE. Lemire [15] proposes a fast stream-processing algorithm $\mathcal{O}(1)$ for causal line SEs. It replaces the line buffers of the previous algorithms by more dedicated memory structure—double-ended FIFO (queue). The author proposed that only locally monotonous signal is necessary for a given operation. The double-ended queue serves well for such a purpose; however, the algorithm works with causal SEs only.

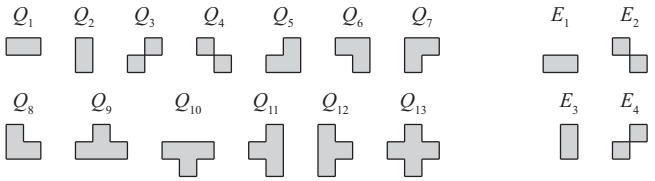
This downside was solved later in Dokládal and Dokládalová [11] who proposed another queue-based algorithm (see Section 4 for further description of his algorithm). The advantages of these queue-based algorithms are low memory requirements, zero latency, and strictly sequential access to data, the paper presents also a comparative study of these features (Table 1).

ii) the decomposition into atomic shapes allows decomposition of large SE into a sequence of small SE, see Sec. 3.1 for more information on SE composition. Xu [30] claims that any 8-convex polygon (convex on 8-connectivity grid, hence 8-convex) is decomposable into a class of 13 non-trivial indecomposable convex polygonal SEs $Q_1 - Q_{13}$ shown in Fig. 1 (a). Normand [19] reduces the class of shapes to only four 2-pixel SEs, see Fig. 1 (b), by allowing the union operator to take place in SE decomposition.

Table 1 Comparison of fast 1-D dilation algorithms.

Algorithm	SE type	Compar. per pixel	Alg. lat.	Data mem.	Working mem.
Naive 1-D	User	$l-1$	0	N	0
HGW	Sym	$3-4/l$	1	N	$2l$
Lemire	Causal	3	0	0	$2l$
Lemmonier	Sym	NC ($\mathcal{O}(1)$)	N	N	N
Dokladal	User	3	0	0	$2l$

Sym = symmetric SE; User = User-defined SE; l = length of a 1-D SE; N = line size; G = number of gray levels; NC = not communicated.



(a) Xu (b) Normand

Fig. 1 Classes of elementary SEs. Any 8-convex polygon SE is decomposable into either: (a) Xu class, or (b) Normand class while using union along with $\oplus$.

For instance, Q_{12} by Xu is obtained as $(E_3 \oplus E_3) \cup E_4$ by Normand.

The drawback of this approach is that large SEs need a long sequence of atomic operations, and may need iterate several times over the image.

For large, rotation-symmetric polygons, the problem has been solved in Soille *et al.* [26], who propose an approximation of polygons by a set of line SEs rotated by different angles. The complete dilation by a polygon still requires several iterations over the image. However, the number of iterations does not depend any longer on the size but rather on the shape. It needs three iterations for a hexagon, and four for an octogon. Each diagonal line is computed by the fast 1-D HGW algorithm oriented by the desired angle. The orientation of the SE is achieved through image partition into discrete lines (parallel, with no overlap), along which the image is processed.

More complex SEs can also be computed directly by dedicated algorithms, see Van Droogenbroeck and Buckley [27].

2.1 Hardware implementations

The hardware implementations of mathematical morphology are often called dataflow architectures in literature as they process an image in stream. They can be classified into three groups: (i) 3×3 neighborhood processors, (ii) partial-result reuse (PRR), and (iii) implementing efficient 1-D algorithm with $\mathcal{O}(1)$.

One of the first 3×3 neighborhood architectures was the texture analyzer [14]. It was optimized for linear and rectangular SE by decomposition into line segments. In [13] the authors devised PIMM1 (Processeur Intégré de Morphologie Mathématique) ASIC that contains one numerical unit for

gray-scale images and 8 binary units. However, the mechanism of computation was not communicated. Ruetz and Brodersen [22] proposed another ASIC chip that separates the supported 3×3 SE into line segments as well. Then only 4 diadic comparisons are necessary to compute the 3×3 SE.

More recently, Velten and Kummert [29] propose another delay-line based architecture for binary images supporting arbitrarily shaped 3×3 SEs. The computation of dilation is realized by OR gates (topology was not communicated, probably a tree of diadic OR gates) achieving good performance, which was further improved by spatial parallelism.

Clienti *et al.* [6] proposes a highly parallel morphological System-on-Chip. It is a set of neighborhood processors PoC optimized for arbitrarily shaped 3×3 SE interconnected in a partially configurable pipeline. The dilation itself is carried out by a tree of diadic $\max$ operators, which is pipelined for better performance.

All previous architectures use the naive method to compute the morphological operations. In order to decrease a number of comparisons for large SEs, the PRR method (name proposed in [5]) takes a partial result of a morphological operation by some neighborhood B_1 in an early stage, delays it, and reuses it later in computation by some other neighborhood B_2 obtaining thus even other, larger B_3.

One of the first PRR architectures for 1-D dilation was proposed in Pitas [21] and improved in Coltuc and Pitas [8]. The principle is based on so-called logarithmic SE decomposition. Even though it reduces a number of comparisons from naive $l-1$ to $\lceil \log_2(l) \rceil$, it restricts a family of possible SE shapes to rectangles only.

The family of SE shapes has been enriched by Chien [5]. The authors presented more general concept of PRR that builds the desired SE by a set of distinct Xu elementary neighborhoods, see Fig. 1 (a). The decomposition process is computed by a dedicated algorithm. As a result, it supports arbitrary 8-convex polygon at the cost of additional comparisons. In [5] the PRR method was implemented as an ASIC chip supporting 5-diameter disk SE. Despite decent performance, the chip lacks possibility to control the shape of the SE.

A similar approach has been published by Déforges *et al.* [10]. Based on Normand SE decomposition [19] (a SE is decomposed into a number of causal 2-pixel SEs, which are applied in sequence or in parallel, see Fig. 1 (b)) and combined with a stream implementation, the authors propose a methodology for pipeline architecture design supporting arbitrary 8-convex SEs. The practical limitation comes from the need to create a long pipeline of atomic modules for large SE. Even thought it is not specified in the paper, such pipeline seems to be dedicated to the given SE shape and size.

Regarding the third group, implementations based on efficient algorithms, there are only two proposals in literature. Clienti *et al.* [7] implemented 1-D HGW and Lemonnier algorithms. In order to avoid a hardware-expensive reverse scan, they devised a ping-pong mirroring buffers that provide reverse-scanned data with minimal latency and memory requirements. The major drawback of the two implementations is incapability of supporting vertical, or 2-D, SE along with horizontal scan data reading.

Prior to this paper, Bartovsky *et al.* [2] reported an efficient parallel design based on the 1-D dilation algorithm [11]. However, it supports rectangular SEs only.

From the paragraphs above we can see that there are few hardware architectures capable of supporting polygonal SEs, and none of them is optimized for polygons. These architectures are usually suitable for small SEs but lose hardware resources efficiency for large SEs. In this paper we propose an architecture primarily dedicated to large polygonal SEs using an efficient algorithm.

2.2 Novelty

Most previous implementations can efficiently compute a single dilation or erosion with large or even arbitrarily-shaped SE using the decomposition into simpler, atomic operations. Provided the atomic operations concatenate, the computation can be efficiently implemented in a pipe. Obviously, more complex or larger shapes will require longer pipes. If the size is a priory unknown (e.g. function of a parameter) the complete pipe cannot be instantiated and one will need to iterate several times over the image, and store intermediate results in the memory.

Moreover, the morphological dilation is never (or rarely) used alone, but rather as a basic brick in filters. Also in a typical application consisting of several stages – for example: i) denoising, ii) object detection, and iii) measures – the dilation can even be used tens of times, with various SEs, from small (denoising) to large ones (object detection). Lastly, in geodesic operations (e.g. morphological reconstruction) the dilation can be used a variable number of times. Even though the resources of a long pipe are not excessive, such a realization lacks polyvalence and flexibility. It will only fit the targeted operator or application.

In this proposition, the atomic operation is the dilation by a large polygon (similarly to [2] with rectangles). Such atomic operation situates at a higher level of abstraction which allowing simpler decompositions, and consequently shorter pipes.

This operator has been embedded in a programmable block (dilation/erosion and the SE size and shape). It also retains all beneficial features of the inherent algorithm [11], e.g. the sequential access to data, zero latency and low memory requirements. These advantages are especially beneficial in more challenging applications, such as ASF filters, that can be computed in one or a few image scans.

3 Basic Notions

Let δ_B, ε_B: $\mathbb{Z}^2 \to \mathbb{R}$ be a dilation and an erosion on gray-scale images, parameterized by a structuring element B, as-

sumed to be flat (i.e., $B \subset \mathbb{Z}^2$) and translation-invariant, defined as

$$\delta_B(f) = \bigvee_{b \in B} f_b \; ; \qquad \varepsilon_B(f) = \bigwedge_{b \in \widehat{B}} f_b \tag{1}$$

The hat $\widehat{\ }$ denotes the transposition of the SE, equal to the set reflection $\widehat{B} = \{x \mid -x \in B\}$, and f_b denotes the translation of the function f by some scalar b. The SE B is equipped with an origin $x \in B$.

The basic concatenation of the dilation and erosion forms other morphological operators. The closing and opening on gray-scale images, φ_B, $\gamma_B \colon \mathbb{Z}^2 \to \mathbb{R}$, parameterized by a structuring element B, are defined as

$$\varphi_B(f) = \varepsilon_B[\delta_B(f)] \; ; \qquad \gamma_B(f) = \delta_B[\varepsilon_B(f)] \tag{2}$$

Furthermore, the concatenation of the closing and opening forms sequential filters, e.g., ASF. The λ-order ASF (referred to as ASF$^\lambda$) is composed of the sequence of λ closings and λ openings with a progressively increasing SE. It starts with either the closing or opening

$$\text{ASF}^\lambda = \varphi^\lambda \gamma^\lambda \varphi^{\lambda-1} \gamma^{\lambda-1} \ldots \varphi^1 \gamma^1 \tag{3}$$

$$\text{ASF}^\lambda = \gamma^\lambda \varphi^\lambda \gamma^{\lambda-1} \varphi^{\lambda-1} \ldots \gamma^1 \varphi^1 \tag{4}$$

Then i.e. the ASF$^\lambda$ starting by closing, can be written as

$$\text{ASF}^\lambda = \delta_{B_\lambda} \varepsilon_{B_\lambda} \varepsilon_{B_\lambda} \delta_{B_\lambda} \delta_{B_{\lambda-1}} \varepsilon_{B_{\lambda-1}} \ldots \varepsilon_{B_1} \delta_{B_1}. \tag{5}$$

The initial number of morphological operators 4λ can be reduced using associativity of dilations and erosions. Hence, every two consecutive dilations or erosions may be merged into one to obtain only $2\lambda + 1$ operators, such as

$$\text{ASF}^\lambda = \delta_{B_\lambda} \varepsilon_{B_\lambda \oplus B_\lambda} \delta_{B_\lambda \oplus B_{\lambda-1}} \ldots \varepsilon_{B_1 \oplus B_1} \delta_{B_1}. \tag{6}$$

3.1 SE Decomposition

The separability of n-D morphological dilation into lower dimensions is a well-known property. The decomposed dilations are then applied in a sequence according to the following equation

$$\delta_R(f) = \delta_{H \oplus V}(f) = \delta_H(\delta_V(f)) \tag{7}$$

where R denotes a rectangle, H and V horizontal and vertical line segments, respectively. This decomposition applies, in general, to convex shapes.

In order to suppress the angular anisotropy of rectangles (note the difference between a side length and a diagonal length), one prefers using circles. Regarding the implementation aspects, circles are often approximated by regular polygons (all sides have the same length) that are easily decomposable, originally described in [1, 30].

A $2n$-top ($n \in \mathbb{N}$) regular polygon SE P_{2n} can be decomposed into a set of n line SEs L_{α_i}

$$P_{2n} = \underbrace{L_{\alpha_1} \oplus \cdots \oplus L_{\alpha_n}}_{n \text{ times}} \tag{8}$$

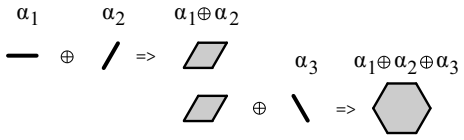
oriented at angle α_i, such as

$$\alpha_i = (i-1)\frac{180^\circ}{n} \;\; [^\circ] \;\; ; i \in \mathbb{N}, i \leq n \tag{9}$$

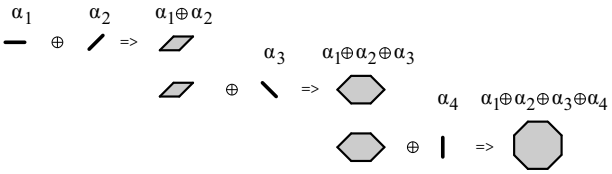
The length of all L_{α_i} is equal to the side of the desired polygon and can be computed from the circumcircle radius R as

$$\|L_{\alpha_i}\| = 2R \sin\left(\frac{180^\circ}{2n}\right) \tag{10}$$

For example, a hexagon can be obtained by three L_{α_i} oriented in $\alpha_i = \{\, 0^\circ, 60^\circ, 120^\circ\}$ on a 6-connected grid, and an octagon by four L_{α_i}, $\alpha_i = \{\, 0^\circ, 45^\circ, 90^\circ, 135^\circ\}$ using an 8-connected grid, see Fig. 2.



(a) Hexagon, $\alpha_i = \{\, 0^\circ, 60^\circ, 120^\circ\}$



(b) Octagon, $\alpha_i = \{\, 0^\circ, 45^\circ, 135^\circ, 90^\circ \,\}$

Fig. 2 Polygon SE composition of line SEs. (a) hexagon is composed of 3 segments, (b) octagon is composed of 4 segments. $\oplus$ operator stands for the Minkowski addition; α_i stands for L_{α_i}.

Hence, from (7) and (8) a 2-D dilation by a $2n$-top polygon $\delta_{P_{2n}}$ of some function $f : \mathbb{R}^2 \to \mathbb{R}$ can be obtained by n consecutive 1-D dilations $\delta_{L\alpha_i}$ by line segments oriented by α_i

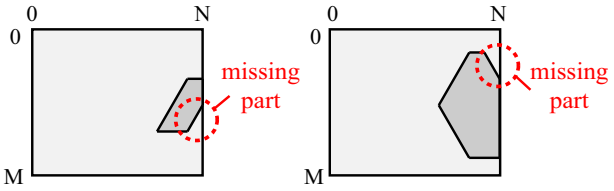
$$\delta_{P_{2n}}(f) = \underbrace{\delta_{L\alpha_1}\,(\,\ldots\;\delta_{L\alpha_n}}_{n \text{ times}}\,(f)). \tag{11}$$

The aforementioned decomposition holds true for the unbounded support $\mathbb{Z}^2$. However, when using real images with a bounded support $D \subset \mathbb{Z}^2$, $D = [1..M] \times [1..N]$, decomposition boundary effects appear if at least one $L_{\alpha_i} \neq \{0^\circ, 90^\circ\}$ is used. The cause is that the Minkowski addition of all decomposed line segments in (8), which are cropped by image boundaries after every L_{α_i} of that concatenation, does not necessarily correspond to P_{2n} cropped by image

boundaries just once as desired. It is expressed by the following expression where $D\cap$ represents intersection with the image support D

$$D \cap (L_{\alpha_1} \oplus \ldots \oplus L_{\alpha_n}) \neq D \cap (L_{\alpha_n} \oplus \ldots \\ \ldots \oplus D \cap (L_{\alpha_2} \oplus D \cap (L_{\alpha_1}))). \tag{12}$$

The illustrative example of such boundary effects with a hexagonal SE is depicted in Fig. 3. We can see that the composition $\alpha_1 \oplus \alpha_2$ is incomplete compared to the desired one in Fig. 2; a small part of the SE is missing. It holds true even for the entire hexagon, the composition $\alpha_1 \oplus \alpha_2 \oplus \alpha_3$ is also incomplete. It is caused by the right boundary cropping not only the final P_{2n}, but also all intermediate results. The cropped values are later missing to form an appropriate polygon section.



(a) $\alpha_1 \oplus \alpha_2$ incomplete (b) $\alpha_1 \oplus \alpha_2 \oplus \alpha_3$ incomplete

Fig. 3 Polygon SE composition without padding. The desired SEs presented in Fig. 2 are incomplete, a small triangle is missing.

This issue is solved by adding a padding to the image. The section of P_{2n} contained inside the image support is then complete, the missing part of P_{2n} is located in the padded area. The added padding contains recessive values, i.e., values that do not affect the computation of a particular morphological operator (for $f{:}D \to V$, $\wedge V$ for dilation, $\vee V$ for erosion). The thickness of the padding is different in the horizontal and vertical direction and is determined by the size of oblique segments, particularly by the half of vertical and horizontal projection

$$B_{\mathrm{H}} = \|L_{\alpha_i}\| \cos(\alpha_2)/2 \quad \text{[pixels]} \tag{13}$$
$$B_{\mathrm{V}} = \|L_{\alpha_i}\| \sin(\alpha_2)/2 \quad \text{[pixels]}. \tag{14}$$

4 Algorithm Description

This section explains the algorithmic principles involved in this paper. First, we expose the 1-D algorithm used for the dilation by line segments with arbitrary orientation. Next, we show how to combine these 1-D computations in order to obtain polygons running in stream. The issue of boundary handling is addressed afterwards.

4.1 1-D Dilation Algorithm

The implementation of (1) consists of searching the extremum of f within the scope of SE B

$$[\delta_B(f)](x) = \max_{b\in B} [f(x-b)] \tag{15}$$
$$[\varepsilon_B(f)](x) = \min_{b\in B} [f(x+b)] \tag{16}$$

The algorithm used below is based on property [11] that for some $B(x)$ (which contains its origin) the computation of the dilation $\delta_B f(x)$ needs only those values of $f(x_i)$ that can "be seen" from x when looking over the topographic profile of f, see Fig. 4. The values shadowed by the mountains - that cannot be maxima - are immediately excluded from the computation.

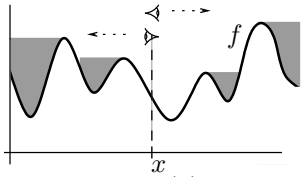


Fig. 4 Computing the dilation $\delta_B f(x)$: Values in valleys shadowed by mountains when looking from x over the topographic relief of f are useless.

From the implementation point of view, assuming a sequential access to the input data f, the dilation $\delta_B f(x)$ depends on points read after x. We say that B is non causal. One can transform a non causal SE to a causal one by utilizing the property that dilation commutes with translation

$$\delta_{B+t} f(x) = \delta_B f(x-t), \qquad \forall t \in D \tag{17}$$

These two principles are used by Alg. 1. For each pixel of some input signal $f : \mathbb{Z} \to R$, the algorithm reads one pixel $F = f(rp)$ at the so-called *reading position rp* and writes back one result pixel $dF = \delta_B f(wp)$ at the current *writing position wp*, such as $rp > wp$. The wp coordinate coincides with the origin of the SE, rp conforms to the most recent input pixel of B. Indeed, the reading position rp is its right-hand side end, which conforms to the intuitive necessity of having read all the samples covered by the SE before computing the dilation.

Alg. 1 is to be called from an outer loop iterating over the writing position in $\delta_B f$, such as *while wp<N*. The writing position wp is to be incremented whenever Alg. 1 outputs a valid value. We give below the pseudocode, for detailed description see [2].

4.2 Stream-Preserving Decomposition of Polygons

Alg. 1 can be used to compute the dilation by L_{α_i} segments in a stream. Its properties make it suitable for composing concatenated operators, namely the sequential access to input and output data, and minimal latency. Therefore, when

Algorithm 1: dF ← 1D_DIL (rp, wp, F, SE1, SE2, N)

Input: F - input signal sample $f(rp)$; rp, wp - reading/writing position; *SE1, SE2* - SE size towards left and right; N - length of the signal
Result: dF - dilated signal sample $\delta_B f(wp)$
Data: Q - Queue (first in, first out)

```
1 while Q.back()[1] ≤ F do
2     Q.dequeue() ;                          // Dequeue useless valuess
3 Q.push({F, rp}) ;                          // Enqueue the current sample
4 if wp − SE1 > Q.front()[2] then
5     Q.pop() ;                              // Delete too old value
6 if rp = min (N, wp + SE2) then
7     return (Q.front()[1] ) ;               // Return valid value
8 else
9     return ({}) ;                          // Return empty
```

the input image is read in a horizontal raster scan mode, i.e., line by line, and every line from the left to the right, the output of Alg. 1 instance conforms to the very same scan order, delayed by some latency defined by the distance between reading rp and writing wp positions. It allows a direct connection of several Alg. 1 instances in a sequence without any need of coupling elements. The resulting 2-D SE is then obtained with minimal latency, that is as soon as all necessary data have been read.

The example of decomposition of a hexagon into three L_{α_i} is depicted in Fig. 5. The image is sequentially read by horizontal L_{0° at the reading position of the polygon (a). The result of the horizontal segment is immediately provided as an input to the first oblique L_{60° at (b) so that the reading position of L_{60° coincides with the writing position of L_{0°. By the very same rule, the result of the L_{60° is brought as input data to the second oblique L_{120° at (c), the writing position of which is the writing position of the complete polygon (d). The total latency is then defined by distance between the reading (a) and the writing position (d) of the polygon.

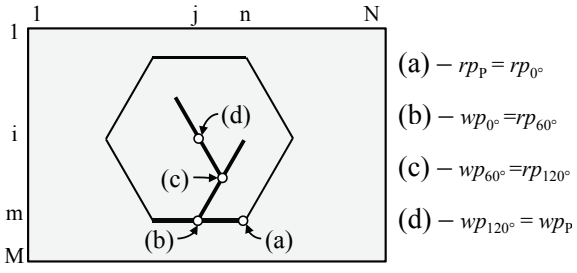


Fig. 5 Stream concatenation of three L_{α_i} into hexagonal SE P; rp/wp - reading/writing position.

The computational complexity of (11) remains almost constant w.r.t. the SE size (except the padding)

$$\mathcal{O}((N + 2B_{\mathrm{H}})(M + 2B_{\mathrm{V}})) \tag{18}$$

for an $N \times M$ image, and B_{H}, B_{V} padding sizes. Provided that of $B_{\mathrm{H}} \ll N$ and $B_{\mathrm{V}} \ll M$, it reaches the complexity of rectangles $\mathcal{O}(NM)$, see [2].

4.3 Discretely Inclined 1-D Segments

The oblique segments included in a hexagon and an octagon, i.e., L_{α_i}, $\alpha_i = 45, 60, 120, 135^\circ$, need appropriate addressing to determine the pixels to process. Note that all inclinations verify $\alpha_i \geq 45^\circ$, and the coefficients k_i verify $k_i = \tan\alpha_i \geq 1$. If we use 8-connectivity for $k_{45^\circ,135^\circ} = \pm 1$, and 6-connectivity for $k_{60^\circ,120^\circ} = \pm 2$, we can very easily generate the pixel adressing - for every inclination - by only modifying the original column index col by an additive constant $line/k_i$ such as

$$col_{\mathrm{shift}} = (col + line/k_i) \bmod (N + 2B_{\mathrm{H}}), \tag{19}$$

where the inclination deviation from the vertical direction $line/k_i$ is called *offset* and changes only with a new image line.

5 Hardware Implementation

In this section, we present hardware implementation (called line unit LU) of Alg. 1 for dilation by L_{α_i} with emphasis on the inclined segment computation. Then, we chain several elementary LU units into a pipeline to form the polygonal processing unit PU. Finally we propose the parallel polygonal unit PPU.

5.1 1-D Algorithm Implementation

Alg. 1 along with the col_{shift} addressing feature is seen as a simple Mealy finite state machine (FSM). This FSM controls all algorithm operations over the queue, rp and wp pointers etc. The state diagram (in Fig. 6) of the algorithm behavior consists especially of 2 main states $\{S1,S2\}$ and one auxiliary state EOL. The basic operation of the direct algorithm implementation can be found in [3, 4]. $S1$ state manages the dequeuing loop and pushing of a new value, code lines 1–3; the $S2$ state handles the deletion of outdated values and returns the result, code lines 4–9.

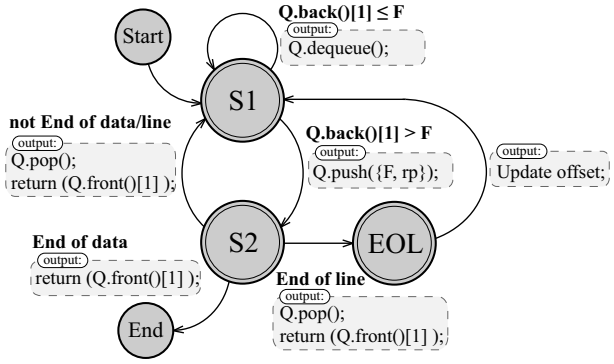


Fig. 6 State diagram of Alg. 1. Conditions of state transitions are typed in bold, output actions are located in gray rectangles.

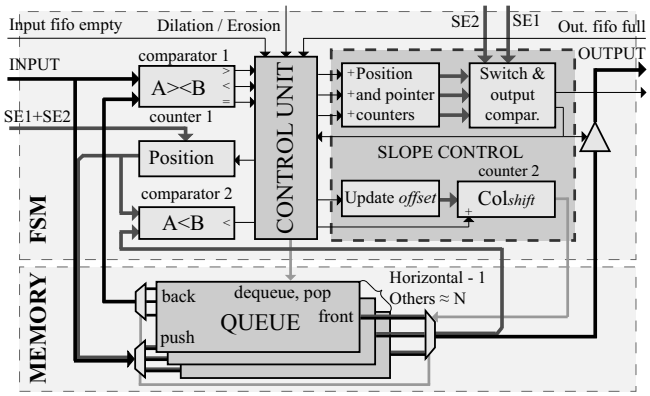


Fig. 7 Overview of the LU architecture. The FSM part manages computation, the memory part contains data storage–queues.

The auxiliary state EOL is entered only at the end of every image line. Its main purpose is to update the offset value, to determine the shifted column addressing. The generation of the necessary inclination is extremely easy since it requires only elementary operations like incrementing, decrementing or stalling.

5.2 1-D Line Unit Architecture

The architecture of the LU unit capable of dilation by different line segments is shown in Fig. 7. The basic description of the preceding version supporting only horizontal and vertical orientation can be found in [4]. Several modifications have been applied to the former version to allow inclined L_{α_i}. We have mainly added the Slope control unit that is highlighted in Fig. 7.

The LU comprises two parts: an FSM part and a memory part containing a collection of double-ended queues. FSM manages the whole computing procedure and temporarily stores values in the memory part. The memory instantiates one queue in the case of horizontal segment, N queues in the vertical case (N is the image width), or $N + 2B_{\mathrm{H}}$ queues in the oblique case. Input and output ports are multiplexed; hence a multiplexor select signal can easily address one queue to work with. The shifted column address (col_{shift}) is used as the select signal.

The processing of one pixel proceeds as follows: In the beginning of $S1$, the last queued pixel is invoked by the Back() operation from the queue and fetched to Comparator 1 where it is compared with the current sample, and dequeued if necessary (code lines 1–2). The current pixel is simply extended with the value of position counter 1 and enqueued (line 3).

The $S2$ invokes the oldest queued pair $\{F, rp\}$ by the Front() operation. This read pixel is a correct result if the set of output conditions (code line 6) is fulfilled. The deleting of an outdated value is managed by comparing the stored position value with the current one in Comparator 2.

The purpose of Slope control is to select the corresponding queue memory which is currently used by Alg. 1. The queues are addressed by the Col_{shift} counter, which is incremented with every pixel of the input image and reset at the end of the image line. The initial reset value of the col_{shift} counter is *offset* (see Section 4.3). The *offset* is updated at the end of every image line (state EOL); its value is incremented or decremented either every line or every other line according to k_i.

5.3 Polygon Unit Architecture

The LU units described above can be arranged in a sequence to form a 2-D Polygon Unit (PU). The overall architecture of the PU unit (see Fig. 8) is composed of three different-purpose parts: computation part, controller, and padding part.

The computation part mainly contains four LUs for distinct L_{α_i} orientations. There are the horizontal unit ($\alpha_1 = 0°$), the first inclined unit ($\alpha_2 = 45°$ or $60°$), the second inclined unit ($\alpha_3 = 135°$ or $120°$), and the vertical unit ($\alpha_4 = 90°$) connected in a simple pipeline; the output of each unit is read by the successive unit which processes the image by further L_{α_i}. The computation part is able to operate either with a hexagon or octagon SE. In the case of the hexagon SE, the vertical unit is bypassed.

Note that the output of every computation unit is an intermediate result image, which can be brought out for another purpose, e.g., a multi-scale analysis descriptor. Then the dilation by line, rectangle, and octagon SEs (all centered) can be obtained during a single image scan (considering units reordering). Only the Remove padding block is to be copied several times for each output data stream.

According to the boundary effects mentioned in Section 4, the inclined units need padding to extend the original image before the processing. The padding is removed after the last 1-D unit. It is carried out by a pair of dual padding blocks at the beginning and the end of the computation part.

The controller ensures the correct global system behavior. It accepts the SE diameter and the shape select signal, then it determines the particular SE sizes for every LU and padding from them, and initiates the computation. The entire set of parameters, i.e., the image width and height, all SE features (size and shape), and the morphological function select, is run-time programmable at the beginning of the frame. These parameters are run-time programmable within the upper bound specified during the synthesis.

For instance, consider the SE size upper bound a 91×91 bounding box, and octagon-capable architecture. This means, the architecture has four LU; each LU supporting at most l=31 pixels segments. During the operation – at the beginning of a frame – the SE can programmed to either of the following: a line up to 31 pixel long, a rectangle up to 31×31 pixels, or an octagon up to 91×91.

To enable processing a uniform input stream, one needs to handle unequal processing rates of LUs. It is caused by variable algorithm latency to compute a dilation for one pixel. Therefore, the balancing FIFO memories are inserted

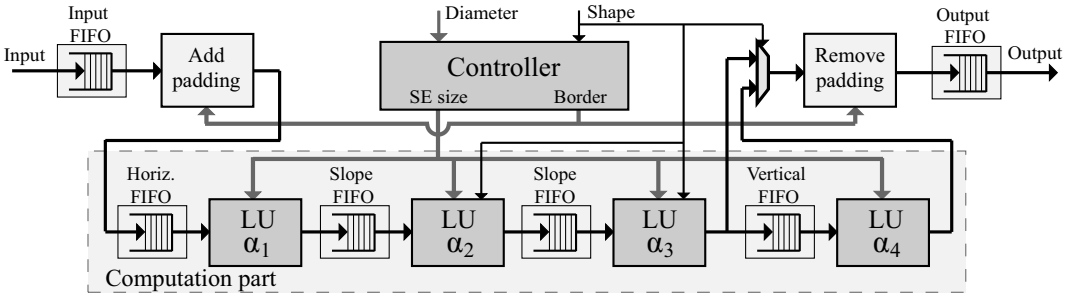


Fig. 8 Overall architecture of the polygonal PU unit. It contains one LU for each $\delta^{\mathrm{L}}_{\alpha_i}$ of (11), control, and padding units.

in front of each 1-D unit, and to the input and output ports. The depth of input and output FIFOs depends on the timing of input data stream (possibility of stalling, synchronization, etc.).

5.4 Memory Requirements

The most significant memory demand is made by the set of queues. Although the algorithm works with separated queues, the queues within each LU are merged into a single dual-port memory, mapped side by side in a linear memory space. Every queue has a related pair of front and back pointers which must be retained throughout the entire computation process in the pointer memory. This approach leads to more efficient implementation.

The LUs have the following memory requirements (considering N×M image including padding, L_{α_i} with bounding boxes $W_x \times H_x$, and bpp bits per pixel):

$$M_{\mathrm{hor}} = W_{\mathrm{H}}(bpp + \lceil \log_2(W_{\mathrm{H}} - 1) \rceil) \quad \text{[bits]} \tag{20}$$

$$M_{\mathrm{ver}} = N((H_{\mathrm{V}} - 1)(bpp + \lceil \log_2(H_{\mathrm{V}} - 1) \rceil) + 2\lceil \log_2(H_{\mathrm{V}} - 1) \rceil) \quad \text{[bits]} \tag{21}$$

$$M_{\mathrm{slope}} = (N + W_{\mathrm{S}})((H_{\mathrm{S}} - 1)(bpp + \lceil \log_2(H_{\mathrm{S}} - 1) \rceil) + 2\lceil \log_2(H_{\mathrm{S}} - 1) \rceil) \quad \text{[bits]} \tag{22}$$

Example: Consider a dilation of 8-bit, SVGA image (i.e., 800×600=$N \times M$) by a hexagon with radius 41 px. Such a SE is decomposed into horizontal SE 21 px wide, and 2 slope SE each 11 px wide and 19 px tall (hexagon SE bounding box is 41×37 px).

The computation memory (the queues) requires (20–22)

$$M_{\mathrm{hor}} = 21(8 + 5) = 273 \quad \text{[bits]}$$

$$M_{\mathrm{slope}} = (811 + 11)((19 - 1)(8 + 5) + 10) = 200'568 \quad \text{[bits]}$$

resulting in total consumption of $M_{\mathrm{all}} = M_{\mathrm{hor}} + 2M_{\mathrm{slope}} \cong$ 392 kbits for the 2-D dilation by hexagon. This is far below the mere size of the image itself $M_{\mathrm{image}} = 800 \times 600 \times 8bpp \cong 3.66$ Mbits which does not need to be stored at any moment.

6 Parallel Implementation

This section describes the Parallel Polygon Unit (PPU) that aims at increasing the computational performance while maintaining as much as possible the beneficial streaming properties of the proposed algorithm.

6.1 Partition of the Image

The parallelism is obtained by use of concurrently working units that simultaneously process different parts of the image (spatial parallelism). The number of instantiated units defines the parallelism degree (*PD*). Since the processing runs in stream, we propose a solution that transforms the input stream into a set of *PD* streams in a way to minimize the waiting-for-data periods of all units. For the sake of clarity, we use *PD*=2 in the description hereafter. A similar method has proven to be useful in [2].

The partition of the input image is twofold, see Fig. 9: an interleaved line-by-line partition for the horizontal α_1 units, and vertical stripes for the vertical and inclined $\alpha_2, \alpha_3, \alpha_4$ units. The final image partition of 2-D image is the intersection of both.

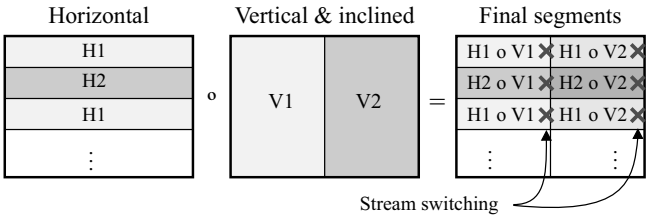


Fig. 9 Example of image partition for *PD*=2: line by line for horizontal orientation; vertical stripes for non-horizontal orientation.

Intuitively, the streams have to be transformed from one type to the other between α_1–α_2, and α_4–*output* in the PU. The transformation is done by simple circular stream switching when a partition edge is encountered. With the beginning of the image, it starts with the H1∘V1 segment on the first line. When the end of this segment is reached, the streams are switched so that segments H1∘V2 (1st line) and H2∘V1 (2nd line) are processed at the same time. Later, it proceeds

to segments H2∘V2 (2nd line) and H1∘V1 (3rd line) and so forth. In general *PD* segments located on a backward diagonal run simultaneously throughout the image (note that the streams are mutually delayed by N/PD pixels).

Processing the partition segments separately introduces undesired border effects on each partition edge. A common solution – similar to padding at image borders – is to introduce an overlap. Contrary to the padding that adds recessive values, the overlap extends a partition by a portion of the neighboring partition. The width of the overlap depends on the size of the SE, and is equal to the width of the horizontal padding B_{H}. Intuitively, the overlap introduces redundant computation, and slightly degrades the performance and minimal latency.

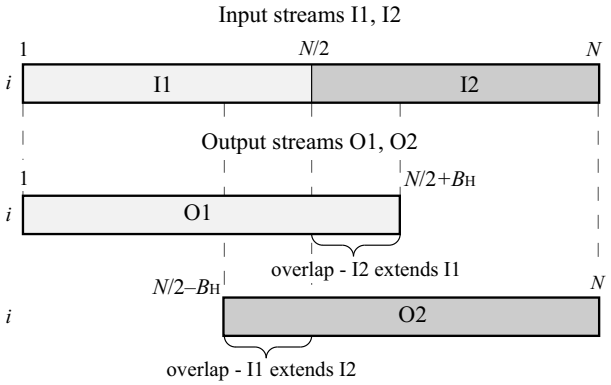


Fig. 11 Addition of overlap on one image line

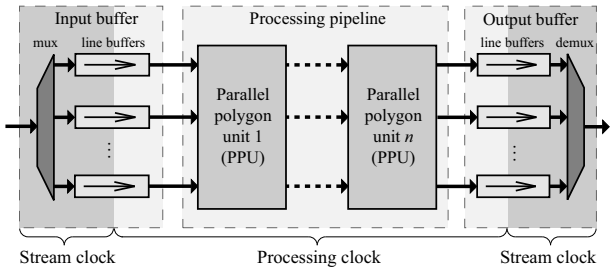


Fig. 12 Overall architecture of parallel ASF application.

6.2 Parallel architecture

At this point, all the previously mentioned principles are brought together to form the Parallel Polygon Unit (PPU). The PPU (see Fig. 10) is scalable with respect to *PD*, the number of parallel streams it can process at the time. Each stream needs one pipeline of four LUs ($\alpha_i, i = 1..4$, just like the PU), two *add overlap* blocks in front of inclined LUs, two *remove overlap* blocks behind inclined LUs, *add padding* at the front end, and *remove padding* at the back end. The PPU also contains a pair of switches to transform the streams from one type to the other, and a controller (ommited in Fig. 10).

Figure 11 shows the introduction of the overlap in the course of the i-th image line. As we know, this line is split into two streams. The streams are labelled I1, I2 before the addition and O1, O2 after (refer also to Fig. 10). The entire I1 stream plus B_{H} pixels of I2 form O1 output stream with overlap, and last B_{H} pixels of I1 and the whole I2 stream form O2.

During the overlap sections, either I1 or I2 stream is mapped to both output streams at the same time. This data duplication does not temporarily allow for parallel processing of both streams and may result in stalling of either stream. However, the effect of the overlap is negligible as long as $B_{\mathrm{H}} \ll N$.

Two important properties are to be noted: (i) input and output streams are mutually delayed by N/PD (ensured by stream switching); (ii) several PPUs can be chained into a pipe. The schematic of some application, e.g., ASF, may look like in Fig. 12. At the front end there is an input buffer transforming the input stream (which is *PD*-times faster than each of *PD* processing streams) into *PD* processing streams Hi ($i = 1..PD$). The transformation only needs i-th image line to be stored in $\{i \bmod PD\}$-th line buffer. In this manner, the processing streams are properly delayed by N/PD pixels. The output buffer transforms *PD* processing streams into one fast stream in the opposite way. One can place as many PPUs as desired between these two buffers in a pipeline or other topology.

The PPU involves the following limitations on the programmability: the image size is set before synthesis, the padding sizes $B_{\mathrm{H}}, B_{\mathrm{V}}$ are computed for the maximal SE, specified before the synthesis. The reason is that handling the varying SE and image sizes would introduce an unreasonable hardware overhead of image partition, padding, and overlap features. The SE features (size and shape) remain fully programmable as in the non-parallel case, see Section 5.3.

7 Experimental results

Hereafter we discuss the results of the proposed implementation. First, we discuss the results of a single 2-D PU and PPU unit, followed by their performance in an application, an ASF filter. We conclude by comparising them with other architectures.

The proposed PU and PPU architecture has been implemented in VHDL and targeted to the Xilinx FPGA Virtex-6 device (XC6VLX240T). The ultimate specification conforms to the following: 8-bit gray-scale images of size up to 1080p (1920×1080 px), height of L_{α_i} up to 31 px (thus, hexagon SE up to 61 px, and octagon SE up to 91 px), and support of uniform stream processing. Notice that all three previous factors affect the memory requirements that (in contrast to the PC), have significant influence on the clock frequency. Our specification implies the clock frequency of 100 MHz.

The timing with respect to (shortly w.r.t.) the image size and the size of the SE (Table 2 and 3) have been evaluated on a natural photo image. We report several measures.

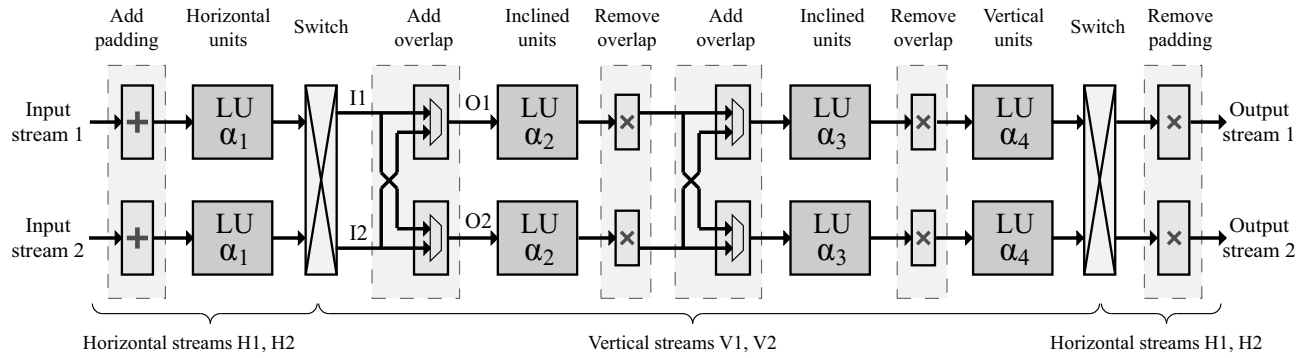


Fig. 10 Overall architecture of the parallel polygonal unit PPU for *PD*=2. The controller and balancing FIFOs are ommited.

Table 2 Timing of PU and PPU w.r.t. image size (SE size = 51 px, *PD*=6).

Image Size	VGA	SVGA	XGA	1080p
Pixel Rate (PU) [clk/px]	2.61	2.53	2.53	2.44
Latency [image line]	25	25	25	25
FPS (PU) [frame/s]	125	82	50	19
FPS (PPU) [frame/s]	599	406	257	105
Speed-up PPU vs. PU [-]	4.79	4.94	5.1	5.34

Table 3 Timing of PU w.r.t. SE size (SVGA image)

SE size [px]	21	31	41	51	61
Rate [clk/px]	2.42	2.46	2.49	2.53	2.58
FPS [frame/s]	85	84	83	82	81
Latency [image line]	10	15	20	25	30

Table 4 Speed-up of PPU w.r.t. *PD* (SVGA image, SE size = 31 px).

Parallelism degree *PD*	2	3	4	5	6
FPS [frame/s]	162	234	306	376	441
Speed-up [-]	1.92	2.77	3.62	4.44	5.22

Table 5 FPGA resources w.r.t. *PD* (SVGA image, SE size = 91 px).

PD	1	2	3	4	5	6
Registers (P)PU	787	1,644	2,469	3,215	4,019	4,850
LUTs (P)PU	2,656	4,831	7,330	9,301	11,540	14,221
Block RAM (P)PU	39	39	59	42	53	63
Registers buf	0	251	355	466	590	671
LUTs buf	0	1,296	1,929	2,545	3,158	3,748

1) The pixel rate gives the average number of clock ticks to process one pixel. It is given by the overall number of clock ticks divided by the image size. One can see that the rate is almost constant w.r.t. both the size of the image and the size of the SE. The slight variation is caused by the size of the SE which affects the size of the padding and overlap, increasing the number of effectively processed pixels, see (18).

2) Latency is expressed in a number of image lines. Note that it is strictly half the SE size. This is a further irreducible factor corresponding to the dependency of the output on the input. This corresponds to the half-height of the SE that needs time to have read enough data to compute the dilation.

3) The last measure is the throughput in terms of the number of frames per second (FPS). The ultimate result we obtain is 105 fps for the 1080p resolution, allowing the 100Hz 1080 FullHD TV standard to be processed in real time.

The speed-up PPU vs PU measures the acceleration obtained from the parallelization. The difference from the ideal upper limit (*PD*=6) is due to the overlap. With increasing image size the acceleration converges towards 6 because of the SE size (and consequently the overlap) becomes negligible with regard to the image size.

Table 4 outlines efficiency of the scalability (that is the parallelism degree *PD*) in terms of the FPS and speed-up. One can see that the real speed-up is somewhat lower than the PD. The difference is due to two factors: (i) the overlap, which demands redundant computation, and (ii) the stream switching that needs inter-stream synchronization which may introduce wait cycles.

Table 5 reveals the cost of parallelization on FPGA resources in terms of registers, LUTs, and BRAMs of the PPU and the pair of input and output buffer as shown in Fig. 12.

(a) Original (b) Filtered

Fig. 13 Example of ASF filtering. A zoom into original and the ASF[3] filtered “Mountain” image.

Table 6 Comparison of several FPGA and ASIC architectures concerning morphological dilation and erosion. N, M stand for the image width and height of respective architectures.

	Processing unit				Hardware System		Application Example ASF^6		
	Technology	Supported SE	Throughput [Mpx/s]	f_{max} [MHz]	Number of units	Supported image	Image scans	FPS [frame/s]	Latency [px]
Clienti [6]	FPGA	arbitrary 3×3	403	100	16	1024×1024	6	66.7	$5NM + 84N$
Chien [5]	ASIC	disc 5×5	190	200	1	720×480	45	12.2	$44NM + 84N$
Déforges [10]	FPGA	arbitrary 8-convex	50	50	1	512×512	13	14.7	$12NM + 84N$
This paper	FPGA	regular polygon	195	100	13	1024×1024	1	185	$84N$

7.1 Alternating Sequential Filter

The ASF filter is an essential method of morphological filtering, see example Fig. 13. Since the ASF is applied as a sequence of alternating dilations and erosions with a changing SE, it can be advantageously implemented by chaining instances of the proposed architecture into a pipeline structure. The output of each operator is immediately processed by a subsequent operator to achieve the following beneficial properties: (i) all the operators are being applied in parallel (temporal parallelism), (ii) the image is filtered with minimal latency inferred by the Minkowski addition of all SEs.

Table 7 illustrates the performance of ASF$^\lambda$ in terms of the experimentally achieved FPS and the inferred latency. Note that the frame rate of the whole ASF decreases with respect to the order λ since larger SE implies larger padding and overlap. However, the performance of the filters is comparable with the rate of a single unit in Table 2.

Table 7 Timing of ASF$^\lambda$; SVGA image size, *PD*=6.

Order of ASF λ	1	2	3	4	5	6
Number of δ, ε	3	5	7	9	11	13
Size of max. SE [px]	5	9	13	17	21	25
FPS by PUs [frame/s]	88	87	86	86	86	85
FPS by PPUs [frame/s]	491	483	466	440	415	387
Latency [image line]	4	12	24	40	60	84

7.2 Architecture Comparison

The implementation and performance comparison of our architecture with the others is outlined in Table 6. At first, we take into account single 2-D units only. Clienti [6] yields a high throughput for an elementary SE 3×3. The Chien [5] ASIC chip achieves a reasonable throughput with a small 5×5 diamond SE. Both architectures use homothecy to obtain larger SEs. On the other hand, one Déforges [10] unit supports various 8-convex SEs in one scan. As mentioned in the state of the art, the programmability of the modules, namely the possibility to control the SE shape after the synthesis is not clear. Smaller throughput is probably caused by usage of a less powerfull device than the rest of implementations.

Lets take as an application example of a compound morphological operator, consider ASF$^6 = \delta_{13\times13}\varepsilon_{25\times25}\ldots\varepsilon_{5\times5}\delta_{3\times3}$ that consists of 13 morphology operations. One Clienti's system instantiates 16 elementary 3×3 processing units. Hence, it will require 6 image scans (the entire image must be stored in the memory). Chien also uses the homothecy, therefore, as many as 45 scans are to be done. In the case of Déforges, neither FPGA occupation with respect to the the size of SE nor possibility of using multiple instances in a single chip was communicated. We consider that only one unit fits the FPGA, so 13 image scans are needed. Of course, if several units fit the FPGA surface, it will reduce proportionally the number of scans. This is true for all streaming architectures.

Obviously, between two consecutive scans the data are read/written from/into the memory that degrades performance and significantly increases latency to orders of several image scans. The dense memory traffic might also overwhelm the data bus.

From the estimated performance results for the ASF^6 in Table 6 we observe that the high use of homothecy tends to increase the number of necessary image scans. Indeed, all Clienti, Chien, and Déforges (a) whose solutions are efficient for small SE sizes and short concatenations become more or less penalized for longer concatenations; their performance drop down with the higher numbers of necessary image scans. On the other hand, Déforges (b) and our work, which does not need more than one image scan, attain the high performance for ASF^6 comparable to the performance of a single 2-D unit.

These features allow a temporal-parallel execution of all atomic operators that is essential to obtain the real-time performance for high demanding applications. In addition, the low memory requirements facilitate embedding several instances of proposed computation units into a single FPGA circuit.

8 Conclusions

It is widely known that the processing data in stream allows to reduce latency, memory consumption and increases the system throughput. Until recently, computing morphological dilations or erosions in stream was only possible for small, limited neighborhoods [5,6,10], or large rectangles [2]. Dilations by large polygons were computed iteratively, by using the homothecy. This required an external memory for intermediate data, limited the flexibility, and drastically increased system latency.

This paper opens the possibility of stream execution to morphological dilation with large polygons. Although the decomposition of polygons into the Minkowski addition of inclined lines has been known for years [1], we bring several suggestions that—combined together—allow the execution in stream.

We show how to implement dilation by inclined linear segments with sequential access to input and output data. We show how to handle border effects, and recall (since this is less known) that it requires large padding. Furthermore, we show how to partition an image to introduce efficient spatial parallelism while maintaining sequential access to data at all levels. This avoids increasing the system clock by dividing a fast data stream into several slower streams to process at a slower rate. We show how to efficiently handle the border effects on the partition.

The proposed polygon decomposition uses sequential access to both input and output data. This allows for temporal parallelism, where in concatenations like $\ldots\delta\varepsilon\delta\ldots$ all these operators run simultaneously on the time-delayed data. We attain a very low (nearly optimal) latency, which has beneficial impacts on memory consumption. No external memory is used even for large SEs and large images. All these aspects brought together allow for a considerable data throughput for sequential morphological filters. We have implemented a programmable IP block, usable in industrial systems running under heavy timing constraints satisfying up to the 100Hz 1080p FullHD TV requirements.

References

1. R. Adams. Radial decomposition of discs and spheres. *CVGIP Graphical models and image processing*, 55(5):325–332, 1993.
2. J. Bartovský, P. Dokládal, E. Dokládalová, and V. Georgiev. Parallel implementation of sequential morphological filters. *Journal of Real-Time Image Processing*, pages 1–13. 10.1007/s11554-011-0226-5.
3. J. Bartovský, P. Dokládal, E. Dokládalová, and V. Georgiev. Stream implementation of serial morphological filters with approximated polygons. In *17th IEEE ICECS*, pages 706–709, Dec. 2010.
4. J. Bartovský, E. Dokládalová, P Dokládal, and V. Georgiev. Pipeline architecture for compound morphological operators. In *IEEE ICIP'10*, pages 3765–3768, Sept. 2010.
5. S.-Y. Chien, S.-Y. Ma, and L.-G. Chen. Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1156 – 1169, Sept. 2005.
6. Ch. Clienti, S. Beucher, and M. Bilodeau. A system on chip dedicated to pipeline neighborhood processing for mathematical morphology. In *EUSIPCO 2008*, Lausanne, Aug. 2008.
7. Ch. Clienti, M. Bilodeau, and S. Beucher. An efficient hardware architecture without line memories for morphological image processing. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 147–156, Berlin, Heidelberg, 2008. Springer-Verlag.
8. D. Coltuc and I. Pitas. On fast running max-min filtering. *IEEE Transactions on Circuits and Systems II*, 44(8):660–663, aug 1997.
9. C. Coster and J.-L. Chermant. Image analysis and mathematical morphology for civil engineering materials. *Cement and Concrete Composites*, 23(2-3):133 – 151, 2001.
10. O. Déforges, N. Normand, and M. Babel. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. *Journal of Real-Time Image Processing*, 2010. DOI: 10.1007/s11554-010-0171-8.
11. P. Dokládal and E. Dokládalová. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22(5):411–420, 2011.
12. J. Gil and M. Werman. Computing 2-D min, median, and max filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):504–507, 1993.
13. J.C. Klein and R. Peyrard. Pimm1, an image processing ASIC based on mathematical morphology. In *ASIC Seminar and Exhibit*, pages P7 –1/1–4, sep 1989.
14. J.C. Klein and J. Serra. The texture analyser. *J. of Microscopy*, 95:349–356, 1972.
15. D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *CoRR*, abs/cs/0610046, 2006.
16. F. Lemonnier and J. Klein. Fast dilation by large 1D structuring elements. In *Proc. Int. Workshop Nonlinear Signal and Img. Proc.*, pages 479–482, Greece, Jun. 1995.
17. G. Matheron. *Random sets and integral geometry*. Wiley New York, 1974.
18. L. Najman and H. Talbot, editors. *Mathematical Morphology: From Theory to Applications*. ISTE Ltd and John Wiley & Sons Inc, 2010.
19. N. Normand. Convex structuring element decomposition for single scan binary mathematical morphology. In *Discrete Geometry for Computer Imagery*, volume 2886 of *LNCS*, pages 154–163. Springer Berlin, Heidelberg, 2003.
20. J Pecht. Speeding-up successive minkowski operations with bit-plane computers. *Pattern Recognition Letters*, 3(2):113 – 117, 1985.
21. I. Pitas. Fast algorithms for running ordering and max/min calculation. *Circuits and Systems, IEEE Transactions on*, 36(6):795–804, June 1989.
22. P.A. Ruetz and R.W. Brodersen. Architectures and design techniques for real-time image-processing IC's. *Solid-State Circuits, IEEE Journal of*, 22(2):233–250, apr 1987.
23. J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.
24. J. Serra. *Image Analysis and Mathematical Morphology*, volume 2. Academic Press, New York, 1988.
25. J. Serra and L. Vincent. An overview of morphological filtering. *Circuits Syst. Signal Process.*, 11(1):47–108, 1992.
26. P. Soille, E. J. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(5):562–567, 1996.
27. M. Van Droogenbroeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recogn. Lett.*, 17(14):1451–1460, 1996.
28. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recogn. Lett.*, 13(7):517–521, 1992.
29. J. Velten and A. Kummert. Implementation of a high-performance hardware architecture for binary morphological image processing operations. In *MWSCAS '04*, volume 2, pages 25–28, 2004.
30. J. Xu. Decomposition of convex polygonal morphological structuring elements into neighborhood subsets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(2):153–162, 1991.

JRTIP manuscript No.
(will be inserted by the editor)

Pavel Karas · Vincent Morard · Jan Bartovský · Thierry Grandpierre · Eva Dokládalová · Petr Matula · Petr Dokládal

GPU Implementation of Linear Morphological Openings with Arbitrary Angle

Received: date / Revised: date

Abstract Linear morphological openings and closings are important non-linear operators from mathematical morphology. In practical applications, many different orientations of digital line segments must typically be considered. In this paper, we (1) review efficient sequential as well as parallel algorithms for the computation of linear openings and closings, (2) compare the performance of CPU implementations of four state-of-the-art algorithms, (3) describe GPU implementations of two recent efficient algorithms allowing arbitrary orientation of the line segments, (4) propose, as the main contribution, an efficient and optimized GPU implementation of linear openings, and (5) compare the performance of all implementations on real images from various applications. From our experimental results, it turned out that the proposed GPU implementation is suitable for applications with large, industrial images, running under severe timing constraints.

Keywords morphology, opening, closing, linear, 1-D SE, parallel, efficient, algorithm, implementation, GPU

1 Introduction

Openings and closings are non-linear image operators of mathematical morphology [1]. They are at the basis of statistical measures called granulometries [2–5] and of a class of non-linear morphological filters called Alternate Sequential Filters (ASF) [6, 7].

In practical applications, the granulometries allow estimation of a priori unknown geometrical characteristics of objects in the image. For illustration, we can cite (1) medical imaging applications e.g., blood cell classification [8], (2) automated document analysis [9], or (3) industrial control [10]. The role of the ASF filters is to reduce the noise while preserving the principal features in the image. They represent the principal element of numerous applications e.g., texture analysis [11] or remote sensing [12]. Even the morphological openings themselves are useful for their filtering properties in some industrial applications such as [13].

Generally speaking, to obtain the desired result–size distribution or filtering effect–we have to use a sequence of openings and/or closings with varying parameters of the applied computing window, so-called structuring element (SE). For a given shape of the SE, these variable parameters are the progressively increasing size of SE and rotation angle. In order to ensure the exhaustivity of the result, applications often require computing of an enormous number of iterations with greater SE, often approaching hundreds of pixels. Considering continually increasing image resolution used in industrial applications, one can intuitively feel that it results in overwhelming requirements on the computing power. This is true even despite recent efficient algorithms [14, 15].

In this context, we study how to efficiently implement the above mentioned operators on graphics cards with the objective to reduce these computing requirements on the system. Initially, graphics cards were designed for graphics purposes only and were not programmable. Based on numerous parallel processors they were very powerful compared to their price. Current GPUs have passed the one Tera FLOPS barrier, and there is no need to use dedicated graphics languages any more since several frameworks have been developed for GPGPU[1] purposes: CUDA [16] by nVidia and OpenCL [17] by the Khronos Group are today

P. Karas
Centre for Biomedical Image Analysis, Faculty of Informatics, Masaryk University, Botanicka 68a, 60200 Brno, Czech Republic. E-mail: xkaras1@fi.muni.cz

V. Morard · P. Matula · P. Dokládal
Centre of Mathematical Morphology, Department Mathematics and Systems, Mines ParisTech, 35, rue St. Honoré, 77300 Fontainebleau Cedex, France. E-mail: {vincent.morard, petr.matula, petr.dokladal}@mines-paristech.fr

P. Matula
University of Heidelberg, BIOQUANT, IPMB, and German Cancer Research Center, Dept. Bioinformatics and Functional Genomics.

J. Bartovský · T. Grandpierre · E. Dokládalová
Laboratoire d'Informatique Gaspard-Monge, Equipe A3SI, Université Paris-Est, ESIEE Paris, 93162 Noisy-le-Grand Cedex, France. E-mail: {j.bartovsky, t.grandpierre, e.dokladalova}@esiee.fr

[1] General-purpose computing on graphics processing units

the most popular in the GPGPU community. Both are based on C language extensions. Notice that in this paper, we use the CUDA language for all presented implementations on GPU. Nevertheless, the principles remain the same when moving to a different language.

In our study, we focus on the morphological openings using linear SE under arbitrary angle, essential in a wide range of practical applications dealing with linear image structures such as fingerprint analysis, geosciences [18], and vessel segmentation [19].

The main novelty of this paper consists of an efficient and optimized GPU implementation of the algorithm by Bartovsky et al. [15], which turned out to be the most suitable for a parallel implementation on GPU. We also present performance comparisons and experimental results of all implementations on real data. It turned out that the proposed GPU implementation can compute linear openings for 180 directions of an image of size 640×640 pixels within 60 ms.We also show that the proposed implementation is significantly faster than the state-of-the-art implementation in the OpenCV_GPU library [20], especially for larger SEs.

In the remainder of this paper we start by the introduction of the mathematical background in Section 2. Section 3 reviews and compares the state of the art of efficient implementations for computing linear morphological openings of different orientations. Section 4 presents the first GPU implementation of two candidate algorithms by Bartovsky [15] and Morard [14]. Afterwards, the Section 5 describes the optimized implementation of Bartovsky algorithm, including the parallelism enhancement discussion. Section 6 illustrates the use of this efficient implementation in practical applications. It includes also the discussion of overall experimental results. Finally, the conclusions recall the main contributions of our work and briefly introduces its perspectives.

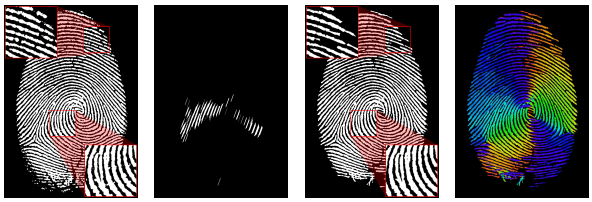
2 Basic Notions

Before defining morphological openings and closings, we define erosions and dilations for gray-scale images. Let a mapping f: $\mathbb{Z}^2 \to \mathbb{R}$ be a gray-scale discrete image, and $\mathscr{F}$ be the collection of all such images. The support used throughout this paper shall be a rectangular subset domain of $\mathbb{Z}^2$.

The erosion and dilation are mappings $\varepsilon_B, \delta_B : \mathscr{F} \to \mathscr{F}$, parameterized by the so-called structuring element (SE) B, $B \subset \mathbb{Z}^2$. Here, we restrict the family of SE to flat and translation-invariant. Hence, the dilation and erosion are, as usually, defined by

$$\delta_B(f) = \bigvee_{b \in B} f_b \,; \qquad \varepsilon_B(f) = \bigwedge_{b \in \widehat{B}} f_b \tag{1}$$

where the hat $\widehat{\ }$ denotes the transposition of the structuring element, i.e., $\widehat{B} = \{x \mid -x \in B\}$, and f_b denotes the translation of the function f by some vector b, and $\bigvee$ and $\bigwedge$ denote the supremum and infimum on a collection of functions.



(a) Input (b) γ_λ^{70} (c) $\vee\gamma_\lambda$ (d) ζ_λ

Fig. 1 Example application of linear openings. **(a)** input image, **(b)** linear opening with a digital line segment of length $\lambda = 41$ pixels and orientation 70 degrees, **(c)** enhancement of linear structures, and **(d)** color coded local orientation of linear structures calculated using definitions in Eq. (3). In **(a)** and **(c)**, two zoomed regions are shown for a better comparison.

Similarly, the morphological openings and closings are mappings γ_B, φ_B: $\mathscr{F} \to \mathscr{F}$, also parameterized by B. The transposition $\widehat{B}$ used in the definition of erosion ensures that the dilation and erosion form a so-called adjunction pair. This allows us to obtain the morphological opening and closing by concatenation of the dilation and erosion:

$$\gamma_B(f) = \delta_B[\varepsilon_B(f)] \,; \qquad \varphi_B(f) = \varepsilon_B[\delta_B(f)]. \tag{2}$$

Erosions and dilations are dual under complementation, i.e. for functions $\delta_B = -\varepsilon_B(-f)$, see e.g. [21]. Consequently, γ_B and φ_B are also dual and all algorithms developed for openings can easily be used also for closings.

In the sequel, we focus on linear morphological openings and closings obtained with SE in a form of a 1-D digital line segment of the length λ and orientation α, denoted by γ_λ^α and φ_λ^α, respectively.

Further operators can be built based on linear openings. The first operator, $\vee\gamma_\lambda(f)$, is computed by taking the supremum of the openings by digital line segments in all orientations; the second operator, $\zeta_\lambda(f)$, can extract the local orientation of linear structures:

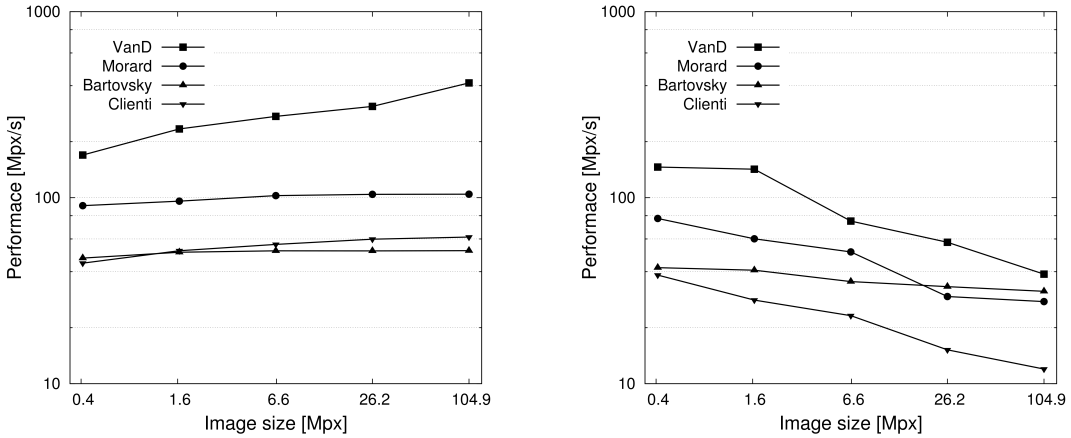
$$\vee\gamma_\lambda(f) = \bigvee_{\alpha \in [0,180]} \gamma_\lambda^\alpha(f) \,; \qquad \zeta_\lambda(f) = \arg \bigvee_{\alpha \in [0,180]} \gamma_\lambda^\alpha(f). \tag{3}$$

As an example, the effect of these operators on a real image is presented in Fig. 1.

3 Opening Algorithms

Opening algorithms can be divided into three classes: erosion and dilation chaining (a two-stage algorithm), direct computation, and algorithms based on connected component tree building.

Two-stage algorithm: computed by using Eq. (2), it is the simplest and historically earlier approach used to compute openings. Noting N the number of pixels of the image



(a) Horizontal SE

(b) Vertical SE

Fig. 2 Comparison of CPU implementations for horizontal and vertical SEs of size approx. 5% of the image width.

and L the size of the SE, the complexity of the naïve approach is $O(N \times L)$. The computational complexity was improved throughout the years to make the algorithm feasible for practical applications. In 1985, Pecht [22] defined a logarithmic decomposition of the SE. This decomposition, which removes most of the redundancy, was further extended by Coltuc [23], reducing the complexity to $O(N \times \log L)$. Later, the complexity was further reduced to linear $O(N)$, hence independent of the size of the structuring element, by Van Herk, and Gil and Werman [24, 25]. The linear algorithm will be referred to as the *HGW algorithm* hereafter. In [26], Soille et al. extended this work to arbitrary-oriented openings. In [27], Clienti et al. improved the HGW algorithm by removing the image backward scanning to reduce latency.

Direct Computation: for the family of openings with 1-D SE, new algorithms were introduced recently to directly compute openings in only one scan of the entire image, preserving the complexity of $O(N)$. In [28], Van Droogenbroeck and Buckley introduced an algorithm based on anchors, allowing very fast computation of the linear openings. The anchors are points, which are not affected by the opening. Nevertheless, the algorithm uses a histogram. The main drawback of using a histogram is the increasing memory consumption for finely quantized data. Even though the memory is no longer an issue for PC architectures, it becomes a penalizing factor for parallelized implementations on other architectures with limited memory like GPU. Secondly, search over a long histogram becomes costly and finely, for floating point accuracy, the histogram can not be used at all.

Later, Morard et al. [14] introduced a very simple algorithm based on an ordered stack of *cords* where a cord refers to a continuous set of pixels of intensity greater than or equal to a certain value I. With the inclusion relation between cords and by analyzing the length of each cord, the computation of linear openings and of linear granulometries is straightforward. Finally, Bartovsky et al. [15] also developed an algorithm to build linear openings. It sequentially scans the signal and erases every peak narrower than SE. Further explanations on these algorithms are given in section 4.

Connected component tree: such approach was described in [29]. The approach is based on building connected components, hence it can be adopted for more complex tasks such as watershed segmentation [30]. The drawback of the algorithm is that the complexity depends on the number of gray levels in the image and requires random access data, consequently it is not adapted for applications running under strict time constraints.

3.1 Parallel Implementations

There are several implementations of HGW algorithm in the literature since it can be easily parallelized. Brambor [31] described a parallel implementation of the HGW algorithm on SIMD architectures. Their implementation was tested on an Intel CPU with the SIMD-SSE2 instruction set. Clienti [27] improved the HGW algorithm and implemented[2] it on a SIMD architecture as well. Domanski et al. [33] used CUDA to implement the HGW algorithm on GPU, achieving 13–33 $\times$ speedup. There are several drawbacks of the algorithm as it computes openings and closings by dilation-erosion chaining, which requires more computations than direct approaches. It also has larger memory requirements [34].

On the contrary, there are few parallel implementations of the component tree algorithms, among which we can cite Wilkinson [35], Menotti-Gomes [36] on multicore, and also Matas [30] on ccNUMA 4-core. They are effectively

[2] available in Fulguro image processing library [32]

so complex that it is difficult to exhibit some parallelism in these complex algorithms.

Finally, in order to improve the computing efficiency by parallel implementation, direct computation algorithms seem to be the best candidates compared to HGW and component-tree algorithms. This is why we focus on this class of algorithms in the remainder of this paper.

3.2 Selection of a Direct Linear Opening Algorithm

We need to select the best sequential algorithm candidate for a parallel implementation leading to the most efficient execution on a GPU platform. As explained above, such an algorithm has to allow arbitrary angles computing using direct linear opening for lower computation and complexity requirements. Hence, there are three algorithms available to benchmark and compare: (1) Van Droogenbroeck [28] referred to as *VanD*, (2) Morard et al. [14] referred to as *Morard* and (3) Bartovsky et al. [15] referred to as *Bartovsky*. To this list, we will add Clienti algorithm [27] although it is not a direct computation based algorithm. Effectively, this is one of the fastest HGW implementation that can consequently give a useful comparative point. It will be referred to as *Clienti*.

For a fair comparison, all these algorithms were implemented using the same image-processing library with the same interface and with all the optimization flags turned on. We used only one core of an Intel Core i7-870 2.93 GHz CPU for this benchmark. These algorithms have been applied on the texture images shown in Fig. 12(a), (b).

Execution performances of the four algorithms for both horizontal and vertical openings are presented in Fig. 2. The performance P is computed as $P = N/t$, where N is the image size and t is the computation time. This measure is consequently independent of the image size. Note, however, that the performance actually does depend on the image size (see e.g. Fig. 2(b)). Each marked value in the plots represents the performance computed from the mean computation of 100 openings. For each image the length of a structuring element was chosen to be equal to 5 % of the image width/height because, in most applications, the length of SE is considerably smaller than the image size.

From the benchmarks we see that for vertical SEs and for large images, the performances significantly decreases for all algorithms, except for the Bartovsky. This is explained by the fact that this algorithm accesses the data sequentially even for vertical structuring elements. It is clear that VanD algorithm is the fastest, followed by Morard, Bartovsky, and Clienti, for both vertical and horizontal orientation. While VanD achieves the best performance, it is nevertheless unsuitable for parallelization on a GPU because of its high memory requirements especially for higher bit depths and arbitrary oriented SEs. In contrast, Morard and Bartovsky algorithms are able to compute openings and closings efficiently regardless the orientation of the SE or the data type precision with minimum memory requirements. Therefore, we selected these two algorithms for parallelization and implementation onto the targeted GPU architecture.

Algorithm 1: Morard algorithm: $G \leftarrow \text{Open1D}(F, \lambda, \text{S})$

```
Input: F – input 1-D signal, λ – size of SE, S – pointer to
       LIFO stack
Result: G – output 1-D signal
Data: S – a stack of triplets (value, position, flag)
initialize S;
for rp ← 0 to |F| − 1 do
    if S.empty() or F[rp] > S.top(value) then
        S.push({F[rp], rp, false});
    else
        while F[rp] < S.top(value) do
            fz ← S.pop();
            if fz(passed) or rp − fz(position) ≥ λ then
                WriteFlatZones(F, G, rp, S, fz);
            process S;
process all zones remaining in S;
```

4 Basic Implementation on GPU

4.1 Morard and Bartovsky Algorithms

In this section, we briefly introduce the Morard and Bartovsky algorithms for computation of morphological openings and closings with a linear SE. The detailed description can be found in [14,15]. Both algorithms are able to compute the operation in O(N) time with respect to the image size and O(1) with respect to the size of SE. The Bartovsky algorithm was originally designed for streaming architectures such as FPGA and hence performs scanning of the input data in sequential order. Nevertheless, it can be simply modified to perform scanning along lines according to the orientation of SE, much like the Morard algorithm. During the scan, intensity and position of each pixel is stored in an auxiliary data structure if necessary. Whereas the Morard algorithm uses the LIFO stack, the Bartovsky algorithm uses the FIFO queue. For arbitrary directions, both algorithms use the Bresenham's lines, as described in [26].

In the Morard algorithm, the image line is scanned for pixels where the intensity changes. Whenever the current pixel intensity is higher than the preceding, the current pixel is pushed to the stack. In the opposite case, the stack is being emptied while necessary. The algorithm needs to store an extra bit for a boolean flag indicating the status of a pixel. The size of the stack is limited only by the size and the bit depth of the image. After processing the stack, the output is written. The outputs are irregular. A pseudo-code is shown in Alg. 1.

In the Bartovsky algorithm, the image line is scanned for so-called peaks. According to a peak configuration, either a pixel is pushed to the queue or the queue is being emptied. The size of the queue is limited by the size of SE. After

Algorithm 2: Bartovsky algorithm: $G \leftarrow \text{Open1D}(F, \lambda, \text{Q})$

Input: F – input 1-D signal, λ – size of SE, Q – pointer to FIFO queue
Result: G – output 1-D signal
Data: Q – a double-end queue of pairs (*value*, *position*)
initialize Q;
for $rp \leftarrow 0$ **to** $|F|-1$ **do**
  while $F[rp] \leq$ Q.back(*value*) **do**
    process Q;
  Q.push($\{F[rp], rp\}$);
  if $rp =$ Q.front(*position*) $+ \lambda$ **then**
    Q.pop();
  if $rp \geq \lambda$ **then**
    $G[rp-\lambda] \leftarrow$ Q.front(*value*);

processing the queue, the output is written. The writes are regular and follow the reads with the well-defined latency corresponding to the size of SE. A pseudo-code is shown in Alg. 2.

4.2 GPU Parallel Architecture

Before we describe basic GPU implementations of the algorithms, we briefly review some properties of the GPU architecture, as shown in Fig. 3.

From the hardware point of view, a GPU consists of hundreds of so-called *streaming multiprocessors (SM)*. Each includes a number of *shader processor (SP)* cores, a number of *registers*, a small *shared memory* providing communication between SPs, and fast *ALU units* for hardware acceleration of transcendental functions. One *global memory* is shared by all SMs and provides a capacity in order of GB and the memory bandwidth in order of 100 GB/s. There are also two additional read-only cached memory spaces accessible by all threads: the *constant* and *texture* memory spaces. They can help programmers to improve the performance of their implementations [37,38]. In some recent GPU architectures, such as FERMI by nVidia [39], the global memory is cached as well.

From the programmer's point of view, every program consists of two parts, a *host code* for CPU, and a *kernel code* for GPU. Before executing a kernel, the host program allocates a memory on GPU and transfers data if necessary. Then a kernel is configured and executed. The configuration defines the number of threads allocated for kernel execution. Generally, the number of threads should be much higher than number of processing units of GPU. This approach allows (1) the proper scaling on various hardware configurations, and (2) hiding the memory latencies. The threads form groups called *blocks* (as in CUDA [37]) or *work-groups* (as in OpenCL [40]), following the hierarchy of SMs and SPs. Synchronization of threads and data sharing is possible within the group only. The threads are executed concurrently in *warps*, usually of 32 threads each.

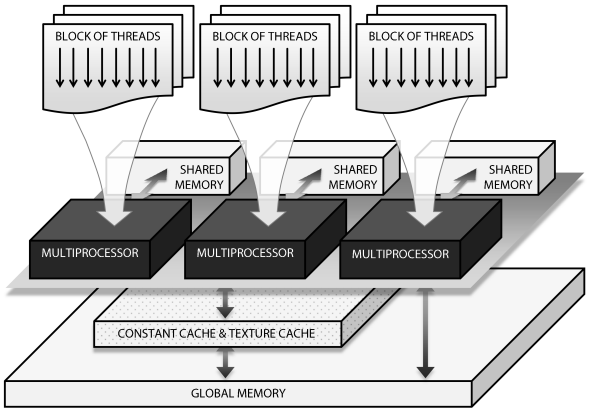


Fig. 3 Thread mapping and memory hierarchy in the GPU architecture.

4.3 GPU Implementation of Morard and Bartovsky Algorithms

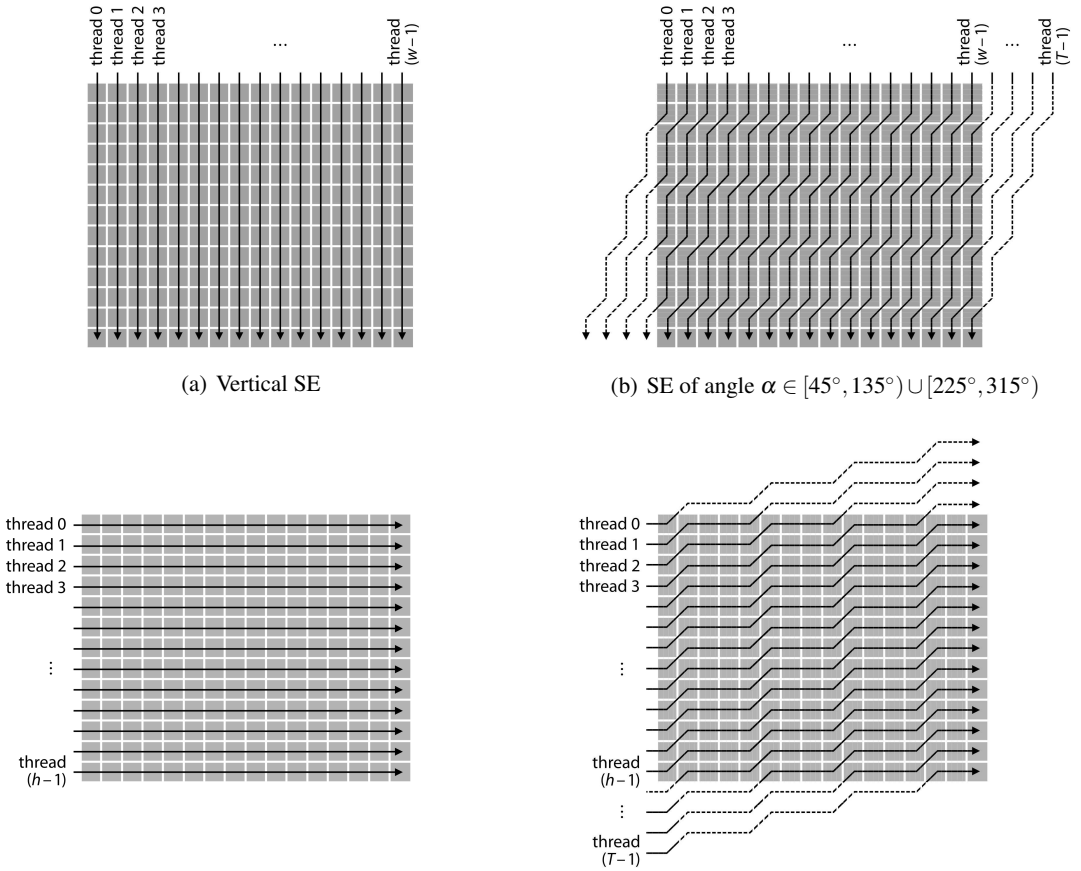
Regarding the design of both algorithms, the input image is scanned in the sequential manner. However, all lines of the image can be scanned concurrently, as shown in Fig. 4. Thus, the parallelism can be introduced by binding individual threads to individual lines of the image. Each image line is processed independently using its own algorithm-dependent stack or queue, respectively. This requires the GPU memory to be large enough to contain all auxiliary data structures. Here, the Bartovsky algorithm is favorable, since the sizes of the queues used are limited to the size of SE whereas the stacks used by Morard algorithm are generally limited by the size of the image.

The mapping of threads for all SE orientations is described in Fig. 4. For an arbitrary angle α, the overall number of threads can be simply computed as follows:

$$T = w + \lceil h|\cot\alpha|\rceil,\ \alpha \in [45^\circ, 135^\circ) \cup [225^\circ, 315^\circ), \quad (4a)$$
$$T = h + \lceil w|\tan\alpha|\rceil,\ \alpha \in [-45^\circ, 45^\circ) \cup [135^\circ, 225^\circ), \quad (4b)$$

where w and h are the width and the height of the input image, respectively. Thus, generally $T > w$ or $T > h$, respectively. Some of threads spend a part of the computation time outside the image domain where they do not compute anything. Thanks to thread locality, this brings little overhead only because in the GPU thread scheduler, thread warps that fall outside the image domain are quickly replaced by those that fall inside.

It should be noted that the processing of both the stack and the queue is data-dependent, thus data accesses are irregular. Therefore, during this part of the computation, threads are divergent. This limits the overall performance of the parallel implementation. In the Bartovsky algorithm, the data accesses to both input and output images are regular.



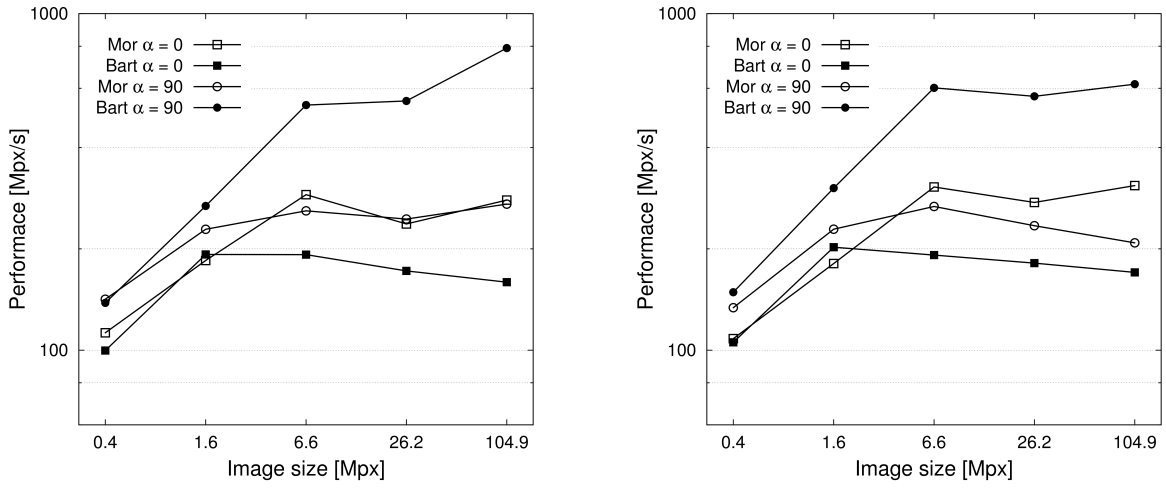
(a) Vertical SE

(b) SE of angle $\alpha \in [45°, 135°) \cup [225°, 315°)$

(c) Horizontal SE

(d) SE of angle $\alpha \in [-45°, 45°) \cup [135°, 225°)$

Fig. 4 The mapping of threads to the 2-D image grid in the GPU implementation of the algorithm for computing 1-D morphological openings/closings. Each thread, denoted by its ID, is mapped to an individual image column or row, respectively.



(a) Image D15

(b) Image D47

Fig. 5 Comparison of basic GPU implementations of Morard and Bartovsky algorithms. Opening was computed with SE of both horizontal and vertical direction and size approx. 5 % of image width.

4.4 Experimental Results

The basic GPU implementations, described in the previous section, were compared on nVidia Tesla C2050 GPU with 14 MPs at 1.15 GHz and 3 GB RAM. Again, all experiments were made with the texture images shown in Fig. 12(a), (b). Results are shown in Fig. 5. The presented performance does not include the times needed for the data transfer between CPU and GPU.

Contrarily to the performance on CPU, the Bartovsky algorithm achieves better performance than the Morard algorithm. However, its performance is highly sensitive to the SE orientation, as shown in Fig. 5. For a horizontal SE, the performance is significantly lower due to extensive L1 cache misses, as detected by Nsight Visual Profiler [41]. As the L1 cache is stored in memory banks and memory accesses are synchronized, most of cache queries are directed to the same memory bank leaving other memory banks unused. This is not the case of the Morard algorithm since it generally keeps the memory accesses irregular.

5 Efficient Implementation of the Bartovsky Algorithm

The choice of the algorithm to optimize is guided by the analysis of the Bartovsky and Morard algorithms in sections 4.3 and 4.4. Despite higher performances of the Morard algorithm on CPU, the Bartovsky algorithm has several advantages:

(1) Both the data accesses to the input and output image are regular. This helps to make the thread execution synchronous and reduces the thread divergence.

(2) The maximum length of the FIFO queue is limited by the length of SE. This strongly limits the memory requirements. Recall that we are to bind one thread per image line (Fig. 4). For large images, with potentionally many threads, it is important to have a small memory footprint of each thread.

In the basic implementation on GPU (contrarily to CPU), the Bartovsky algorithm is sensitive to the SE orientation. Hence, we shall introduce several optimization steps not only to increase the overall performance but, particularly, to keep the performance stable for all orientations. These steps are described in the following sections.

To prove the choice of the Bartovsky algorithm, we applied the optimization steps also on the Morard algorithm and performed tests to compare both optimized GPU implementations. The results are shown in Section 6.

5.1 Parallelism Enhancement

In the basic implementation, the parallelism was introduced by mapping GPU threads to individual image rows or columns, creating the grid of h or w threads, respectively (where h and w are height and width of a 2-D input image, respectively). However, if an input image is not large enough, the GPU's MPs are not fully occupied. Therefore, we introduce more parallelism by splitting the image into two or more parts. In the following text, we refer to them as *zones*. As each pixel can be affected by $(2\lambda-2)$ neighboring pixels (where λ is the length of SE), the zones need to overlap by $2(\lambda-1)$ pixels. An example of the image split for a vertical SE is shown in Fig. 6.

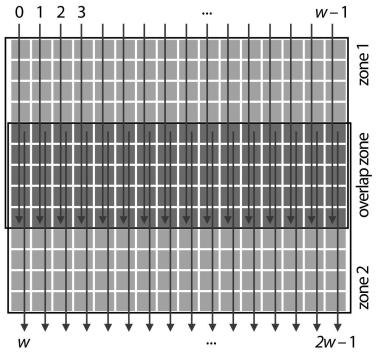


Fig. 6 Image split applied for opening/closing with vertical SE of size 2. Image is split into 2 zones, introducing twice more threads. The threads are denoted by vertical arrows, analogously to Fig. 4.

It is evident that the theoretical speed-up s that can be achieved depends on the SE length λ, the image size and the number of zones Z. For vertical SE we get the following:

$$s = Z\left[1 - \frac{(Z-1)(2\lambda-1)}{h}\right], \tag{5}$$

where h is the image height. For small λ we get $s \approx Z$, while for $\lambda \approx h/(2Z)$ we get $s \approx 1$. It should be noted that for large input images it is not necessary to introduce more parallelism by splitting the image, it can even decrease the performance. The optimal choice of the parameter Z is discussed in section 5.4.

5.2 Optimization of Data Accesses

To reduce the memory latency, the usage of the global memory should be avoided where possible. Alternative memory spaces can be used for both the input image and the FIFO queues. The input image can beneficially be stored in the read-only cached texture memory. This is true also for the recent FERMI architecture, which introduced L1 cache [39, 42], because the texture memory helps significantly improve the performance for horizontal SEs where L1 cache fails due to bank conflicts. The FIFOs can be stored in the shared memory. As the amount of the available shared memory is limited, the maximum length l_{max} of the FIFO is limited to $l_{max} = S/(S_\mathrm{b} \times d)$, where S is the amount of the shared memory available, S_b is the number of threads per block (workgroup) sharing the memory, and d is the size of the data type. Hence, the most recent values of the FIFO are stored in the shared memory in a circular buffer so the position of the first element varies during the computation. The rest of the FIFO

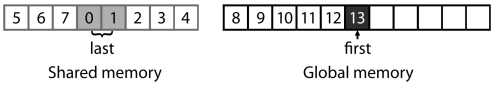


Fig. 7 FIFO storage in the shared and the global memory. The numbers indicate the order of pushing the element into the FIFO. The marked elements (the first one and the last two) are also cached in registers.

is stored in the global memory. The FIFO storage is shown in Fig. 7.

The constraint on the FIFO length mentioned above can be considered as a hard limit. However, using as much shared memory as possible can lead to a performance decrease because in that case, the shared memory is allocated for one thread block per MP only. Since the MPs are capable of execution of more thread blocks, the optimal FIFO length needs to be chosen carefully, as discussed further.

To conclude our choice of memory spaces, using L1 and L2 caches for input data turned out to be inefficient for some SE orientations, as explained in Section 4.4. The texture memory is read-only, cached, and can be allocated up to the size of the device memory. Therefore, it is suitable for the input image. The size of the texture cache is relatively small (8 kB per multiprocessor) but sufficient, thanks to the spatial locality. The shared memory is rewritable, relatively small (up to 48 kB per MP), and fast. In terms of latency, it is comparable with registers, provided that block conflicts are avoided—which is guaranteed in our implementation. Thus, it is suitable for FIFO caching.

5.3 Consideration of Arbitrary Orientations

As noted in section 4.4, the performance of Bartovsky algorithm on GPU is sensitive to SE orientation. Thus, a careful attention should be paid to this issue when computing openings in arbitrary directions. By using the texture memory, the input image is cached and the latency of memory reads is reduced. The final step is to optimize the memory writes to the output image. Experiments proved that openings with SEs of orientation $\alpha \in (-45^\circ, 45^\circ)$ are computed faster when writing the output image transposed. The output image is subsequently corrected using the modified *transpose* kernel from [43].

5.4 Configuration of Performance Parameters

In the optimized GPU implementation, we have introduced several parameters that influence the performance: the block size (work-group size) S_b, the number of zones Z, and the optimal length l of the FIFO buffer allocated in the shared memory. They all depend on these properties: the image dimensions (w or h), the SE length λ, the number of the MPs N_MP available on the used device, the shared memory size per MP S, the warp size W and the number of blocks (work-groups) N_b that can be executed on a single MP. For optimal configuration, the following set of rules should be satisfied:

1. S_b should be multiple of W,
2. $Z \geq (S_\mathrm{b} \times N_\mathrm{b} \times N_\mathrm{MP})/T$ where T is defined in Eq. (4),
3. $Z \leq h/(2\lambda)$ or $Z \leq w/(2\lambda)$ following Eq. (5),
4. $l \leq S/(S_\mathrm{b} \times d \times N_\mathrm{b})$,
5. l should be a power of two allowing the bitwise operator "&" to be used instead of the costly modulo operator for addressing the FIFO queue items.

It should be noted that some of the rules cannot be satisfied in some cases. In particular, rules (2) and (3) can be conflicting for small input images. Performances for some parameter configurations are presented in the following section. CUDA programmers are advised to use a tool called "CUDA Occupancy Calculator" which can help to compute the optimal kernel configuration [43].

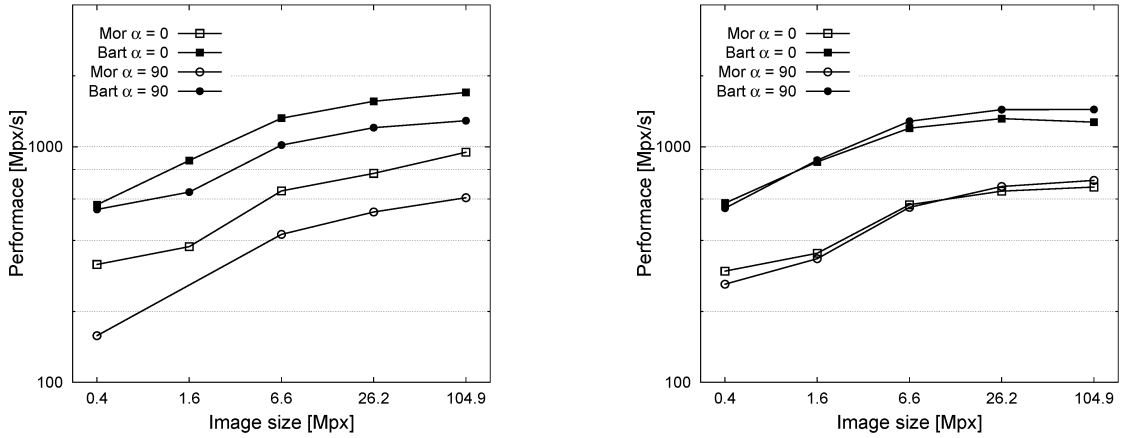
6 Experimental Results

We made the performance analysis of the optimized GPU implementations, based on images with clear linear structures (see Fig. 12), and the results of our comparisons are shown in Fig. 8, 9, and 10.

In the first experiment (Fig. 8), the comparison of optimized GPU implementations of Bartovsky and Morard algorithms is shown, in analogy to Fig. 5. We assumed that the former is more suitable for the GPU architecture than the latter. This assumption was confirmed by the experiments. Thus, in the following experiments, we used the more successful implementation.

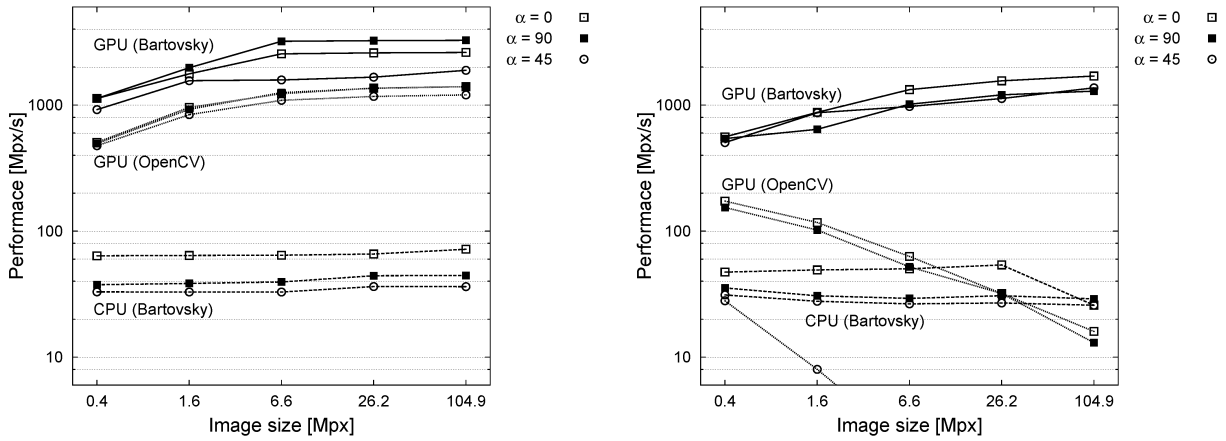
In the second experiment (Fig. 9 and 10), we compared the performance of our GPU implementation, referred to as "GPU (Bartovsky)", to the corresponding CPU implementation, referred to as "CPU (Bartovsky)", and also to the state-of-the-art implementation in the OpenCV library with the GPU support (so-called OpenCV_GPU) [20], referred to as "GPU (OpenCV)". It turns out that our GPU implementation is approximately 10–50 $\times$ faster than the CPU implementation, depending on the input data size and the length of the SE. Despite the fact that the Bartovsky algorithm itself is sequential so the parallelism introduced in its GPU implementation is limited, our implementation is faster than the OpenCV_GPU in every case. Whereas for small SEs the difference between the two GPU implementations is negligible, for larger horizontal and vertical SEs the speedup is up to 50 $\times$. For diagonal (and arbitrarily oriented) SEs, the performance of the OpenCV_GPU implementation falls down very quickly. This is because this implementation uses the NVIDIA Parallel Primitives (NPP) library [44] which supports only simple SE shapes, therefore the line SE has to be represented by a corresponding 2-D rectangle, i.e. a matrix with elements correctly set to 0 or 1.

The most important performance limit of our implementation is the number of threads that can be executed. If the



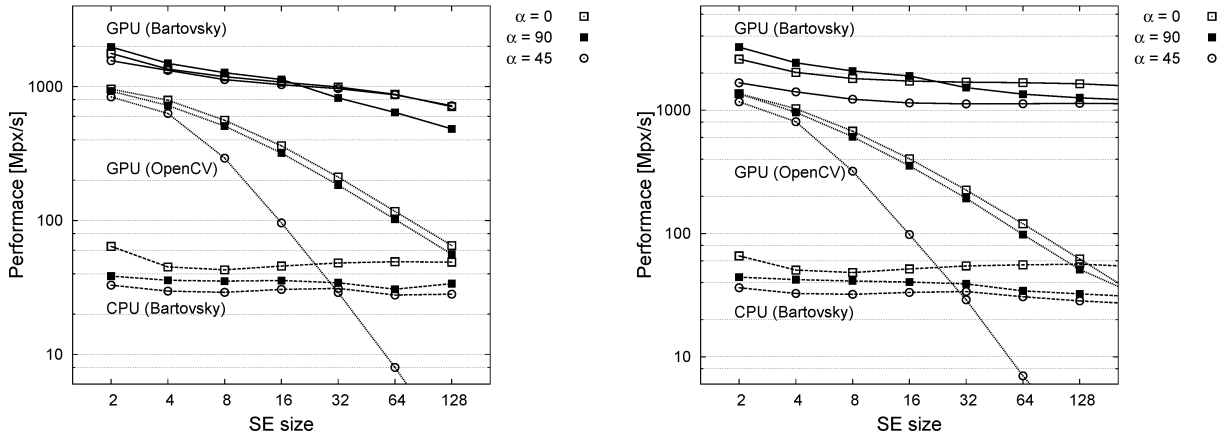
(a) Image D15 (b) Image D47

Fig. 8 Comparison of optimized GPU implementations of Morard and Bartovsky algorithms. For comparison with the basic GPU implementation, refer to Fig. 5.



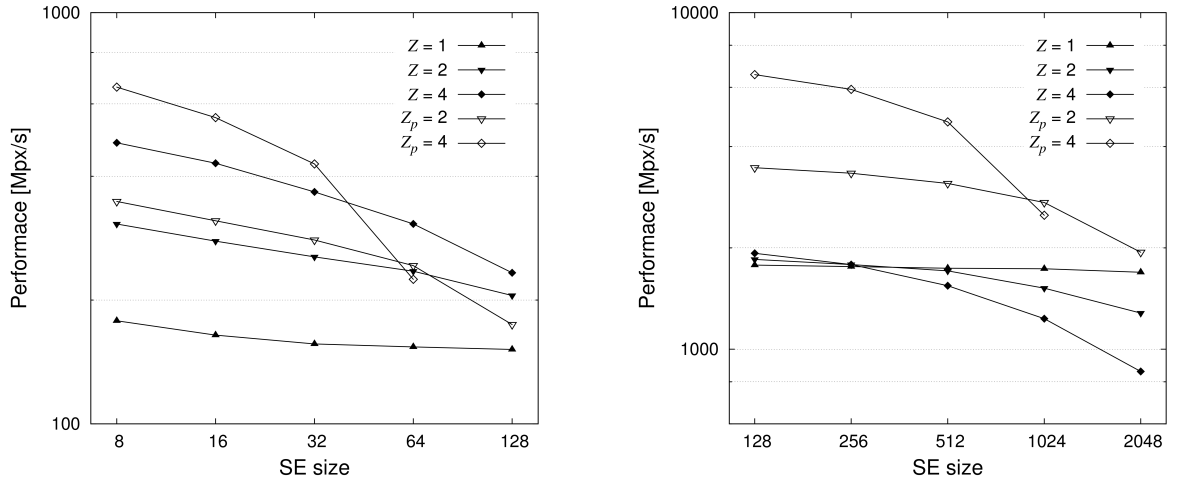
(a) Image D15, SE size = 2 (b) Image D15, SE size = 5 % of image width

Fig. 9 Comparison of CPU and GPU implementations for various image sizes, various SE orientations, and fixed SE sizes.



(a) Image D15, 1280 × 1280 px (b) Image D15, 5120 × 5120 px

Fig. 10 Comparison of CPU and GPU implementations for fixed image sizes, various SE orientations, and various SE sizes.



(a) Image D15, 640×640 px

(b) Image D15, 10240×10240 px

Fig. 11 The contribution of the image split into performance for various image sizes: $Z = 1$ means no split, $Z = 2$, $Z = 4$ means split into 2 and 4 zones, respectively, as described in 5.1; $Z_p = 2$, $Z_p = 4$ means theoretical speedup, as predicted in Eq. (5).

Table 1 Optimal kernel configuration for nVidia graphics cards and for a vertical SE. Choice of the optimal block size S_b, the Z parameter, and the size of the shared memory l depend on many parameters, as described in section 5.4. The Z parameter should be decreased if the SE size is too large. Compute capability refers to a core architecture of nVidia graphics cards [45].

Compute capability 1.x				Compute capability 2.x			
S_b	Image width w	Z	l	S_b	Image width w	Z	l
32	$< 64N_\mathrm{MP}$	4	16	32	$< 64N_\mathrm{MP}$	4	32
64	$< 128N_\mathrm{MP}$	4	8	96	$< 192N_\mathrm{MP}$	4	16
64	$< 256N_\mathrm{MP}$	2	8	96	$< 384N_\mathrm{MP}$	2	16
64	$\geq 256N_\mathrm{MP}$	1	8	96	$\geq 384N_\mathrm{MP}$	1	16

input image is too small, there is not enough threads and the GPU's is underused. This can be avoided by splitting the image in zones (refer to Fig. 6). Adjacent zones need to overlap to avoid border effects. For large SE sizes, the overlap becomes large, with the consequence that the performances decrease, see Fig. 10(a). For larger image sizes, the decrease is proportionally lesser, see Fig. 10(b).

Thanks to the optimizations described in section 5.3 we achieved stable performance for all SE orientations. The performance was tested for all $\alpha \in [0^\circ, 180^\circ)$, although for the sake of simplicity, the graphs show only three orientations. To conclude, our GPU implementation can be used for an arbitrary SE length and direction, achieving the performance more than 1000 Mpx/s. This allows computing one opening on a 40 Mpx image with any 1-D SE in any orientation.

The contribution of splitting the image into zones is shown in Fig. 11 and compared with a theoretical speedup, as computed by Eq. (5). For smaller images, the optimization increases the performance as expected. Note that for large SE sizes, the performance does not decrease as quickly as theoretically predicted. It is because the further parallelism introduced by the split not only occupies more MPs but it also helps to hide memory latencies. For larger images, the number of threads is large enough to occupy MPs, hence the image split does not introduce a further speedup.

The optimal choice of parameters for nVidia graphics cards and for a vertical SE is shown in Table 1. For AMD Radeon graphics cards, the optimum values may differ. For other orientations, the optimal parameters are analogous.

The performance values do not include the data transfers between CPU and GPU. According to our measurements, the time needed to transfer data is comparable with the time needed for the computation of a single opening, hence the overall speedup is half. Here, the GPU implementation is favorable for images larger than 6 Mpx. In the computation of multiple openings, the data transfer overhead is negligible. The performance values for our GPU implementation were obtained on top-class Tesla C2050 GPU (current price 1500 EUR), but we did several test also on $10\times$ cheaper GeForce GTX 470 GPU, and the results were comparable.

6.1 Practical Applications

In practice, linear openings and closings can be used for the detection of either local or global orientations of linear structures. Hence, we tested and compared two CPU implementations (Bartovsky, Morard) and our GPU implementation of linear openings and closings based on images from three different application domains, namely fingerprint analysis, texture characterization, and document analysis. In all cases, we computed a set of linear openings allowing massive parallelism on GPU simply by computing linear openings in all

Table 2 Comparison of CPU and GPU implementations for computation of an angular spectrum. The speedup is computed by comparing the GPU implementation with the best CPU (Morard) implementation.

Image	Image size	No. of angles	Time [ms] CPU (Bart)	CPU (Mor)	GPU (Bart)	Speedup GPU/CPU
Fingerprint	784×1133	180	3768.7	3129.2	115.9	27.0
D15	640×640	180	2425.8	2014.0	58.0	34.7
D47	640×640	180	2397.7	2014.0	55.1	36.6
Music1	1000×1411	81	5830.0	4840.7	129.3	37.4
Music2	1000×1301	81	7297.1	4958.8	136.5	36.3

directions concurrently avoiding the problem with insufficient MPs occupancy. We will show that this leads to a significant speed-up even for small input images. The benchmark results for all applications are shown in Table 2.

6.1.1 Local Orientation of Linear Structures in Fingerprint Images

The first application was the computation of local orientation of linear structures in a fingerprint image (Fig. 1). The operator $\zeta_\lambda(f)$ was computed according to Eq. (3). The GPU implementation achieves a significant speedup (approximately $27\times$) even for small input images (0.9 megapixels).

6.1.2 Angular Spectrum of Texture Images

The second application was the computation of the angular spectrum of texture images. The spectrum was computed in order to find the most important orientation(s) of linear structures in the image. Two test images along with their spectra are shown in Fig. 12. A spectrum $\sigma_\lambda(\alpha)$ of an image f is computed as follows:

$$\sigma_\lambda(\alpha) = \sum_{x \in \Omega(f)} \left[\gamma_\lambda^\alpha(f)\right](x). \tag{6}$$

Again, the GPU implementation achieves a significant speedup (approximately $35\times$) for input images of 0.4 megapixels.

6.1.3 Rotation Detection of Music Sheet Scans

In the third practical application, we detected the rotation of music sheet scans. The music sheets were scanned and stored in an electronic archive. In the process of scanning, a random rotation could occur due to imperfect insertion of the paper to the scanner. The rotation was detected by computing linear closings with large SEs ($\lambda = 250$) of 81 different orientations within the angular range from -10° to 10° with the step of 0.25°. The angular spectrum was computed according to Eq. (6). Two test images along with their spectra are shown in Fig. 13. In this case, the achieved speed-up was approximately $37\times$.

7 Perspectives - Extention to 2-D

The 2-D opening is not separable into two orthogonal 1-D openings as is the dilation. Hence, one cannot directly combine two orthogonal 1-D openings to obtain a 2-D opening.

It is known, that efficient GPU accelerations can only be obtained with simple, regular threads, using as low memory as possible. Hence, the separability principle is useful. Following this idea, one can form 2-D openings by concatenating 2-D erosion and 2-D dilation which are separable. A 1-D dilation algorithm with alike properties as the Bartovsky algorithm has been published in [34].

Assume a 2-D rectangular B in Eq. (1). It can be decomposed into two (horizontal and vertical) 1-D dilations and two 1-D erosions. Similarly, for hexagons we need to compute three erosions and three dilations, and for octagons four erosions and four dilations.

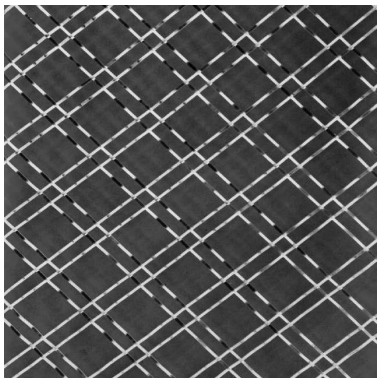
All these orthogonal operators need to be computed sequentially. One cannot expect to obtain the same performances in 2-D as with 1-D openings, since the execution times of the orthogonal operators are added together.

8 Conclusions

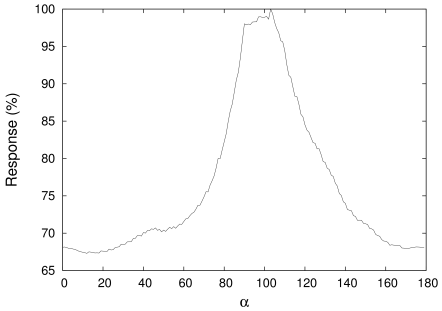
We have reviewed and compared the most efficient linear morphological opening/closing algorithms. At present, the fastest approaches (Van Droogenbroeck and Buckley, Morard et al., and Bartovsky et al.) compute the opening within a single image scan. The algorithm of Van Droogenbroeck and Buckley is the fastest one on CPU, however, it is efficient for 8-bit gray-scale images and for vertical and horizontal linear openings only. Morard and Bartovsky algorithms are applicable to any data accuracy (including floating point).

As described in the paper, both Morard and Bartovsky algorithms themselves are sequential. Hence, the only possibility of introducing more parallelism is on the thread execution level. We explain the choice of the algorithm (Bartovsky) to implement with regard to the GPU architecture (little memory, synchronous execution of threads). We have used various optimization techniques to speed up the code. Mapping various types of data (input, output and FIFOs) to various memory spaces is a crucial aspect. The choice of

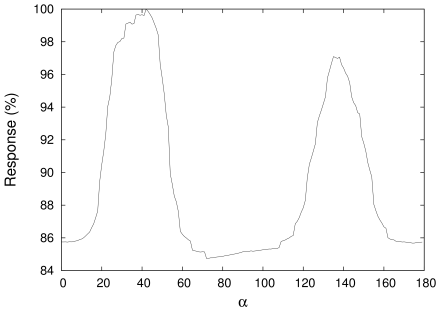
(a) Image D15



(b) Image D47

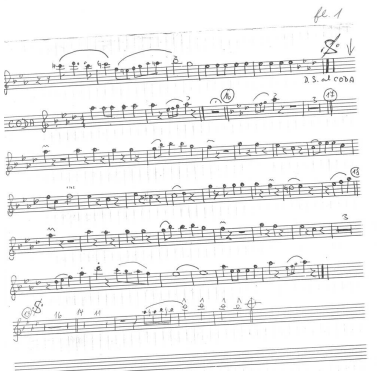


(c) D15 Spectrum $\sigma_{40}(\alpha)$

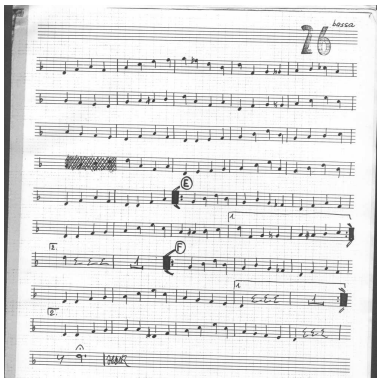


(d) D47 Spectrum $\sigma_{20}(\alpha)$

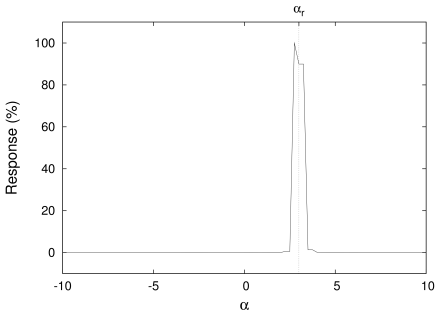
Fig. 12 Texture images [46] and their angular spectra.



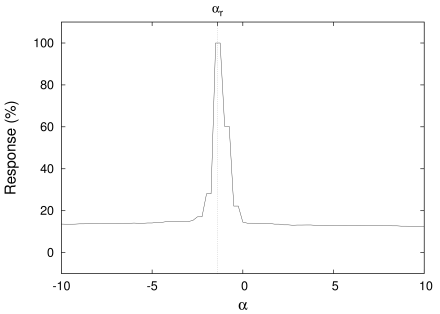
(a) Image Music1



(b) Image Music2



(c) Music1 Spectrum $\sigma_{250}(\alpha)$



(d) Music2 Spectrum $\sigma_{250}(\alpha)$

Fig. 13 Music sheet scans (courtesy of Josef Pilný, Big Band Lanškroun) and their angular spectra. The real (manually measured) rotation angle is denoted by α_r.

memory spaces is described in detail in Section 5.2. We also observed performance limits on small images, hence we try to introduce more spatial parallelism by splitting small images. Finally, we give rules of optimal configuration regarding input image size and the features of the available GPU.

We provide comparisons with the fastest implementations on CPU and on GPU. The current state-of-the-art standard, OpenCV_GPU, is suitable for GPU architecture but has time complexity dependent on the SE size. The proposed implementation obtained stable performance over all orientations and sizes of the structuring element. For a single opening of a large image, we have measured up to 50 $\times$ speedup compared with CPU. For small images, the gain is significant (up to 37 $\times$ speedup) if one computes a set of openings in multiple directions. For example, within 60 ms, the GPU implementation is capable of computing a single opening at arbitrary angle of a 60 Mpx image, or a set of openings in 180 directions of a 640$\times$640 px image.

To conclude, this solution is suitable for applications with large, industrial images, running under severe timing constraints, such as production control in e.g. metallurgy or textile industry. A typical such application requiring using high-resolution images, and running under severe time constraints is the surface control. Thin (often μm) cracks in large surfaces require using high resolution images, and the timing is given by the industrial cycle.

Source codes of the CPU and GPU implementations of the Bartovsky algorithm are publicly accessible under GNU GPL license [47].

Acknowledgements

This work has been done during ERASMUS stage of Pavel Karas at ESIEE Paris. The presented work is the result of collaboration between ESIEE Paris (A3SI team), Mines-ParisTech (CMM) and Masaryk University in Brno (CBIA).

This stay period of Pavel Karas at ESIEE Paris has been also supported by the Ministry of Education, Youth and Sport of the Czech Republic (Projects No. MSM-0021622419 and No. LC535). This research was also supported by the Grant Agency of the Czech Republic (Grant No. P302/12/G157).

References

1. J. Serra. Morphological filtering: An overview. *Signal Processing*, 38:3–11, 1994.
2. L. Vincent. Fast opening functions and morphological granulometries. In *SPIE*, volume 2300, pages 253–267, 1994.
3. S. Batman, E. R. Dougherty, and F. Sand. Heterogeneous morphological granulometries. *Pattern Recognition*, 33(6):1047 – 1057, 2000.
4. L. Vincent. Granulometries and opening trees. *Fundam. Inf.*, 41:57–90, January 2000.
5. E. R. Urbach, J. B. T. M. Roerdink, and M. H. F. Wilkinson. Connected rotation-invariant size-shape granulometries. *Pattern Recognition, International Conference on*, 1:688–691, 2004.
6. J. Serra and L. Vincent. An overview of morphological filtering. *Circuits, Systems and Signal Processing*, 11(1):47–108, 1992.
7. H. Heijmans. A new class of alternating sequential filters. In *I of Proceedings of 1995 IEEE Workshop on Nonlinear Signal and Image Processing*, pages 3–0, 1995.
8. N. Theera-Umpon and P. D. Gader. Counting white blood cells using morphological granulometries. *J. Electronic Imaging*, pages 170–177, 2000.
9. A. Bagdanov and M. Worring. Granulometric analysis of document images. In *Proceedings of the International Conference on Pattern Recognition*, volume I, pages 478 – 481, 2003.
10. S. Outal, D. Jeulin, and J. Schleifer. A new method for estimating the 3d size-distribution curve of fragmented rocks out of 2d images. *Image Analysis and Stereology*, 27(2), 2011.
11. D. Talukdar and R. Acharya. Estimation of fractal dimension using alternating sequential filters. In *Proceedings of the 1995 International Conference on Image Processing (Vol. 1)*, ICIP '95, Washington, DC, USA, 1995. IEEE Computer Society.
12. S. O. Sigurjonsson, J. A. Benediktsson, and J. R. Sveinsson. Street tracking based on sar data from urban areas. In *International Geoscience and Remote Sensing Symposium*, pages 1273–1276, 2005.
13. M. Kowalczyk, P. Koza, P. Kupidura, and J. Marciniak. Application of mathematical morphology operations for simplification and improvement of correlation of images in close-range photogrammetry. In *ISPRS08*, page B5: 153 ff, 2008.
14. V. Morard, P. Dokládal, and E. Decencière. Linear openings in arbitrary orientation in O(1) per pixel. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1457–1460, May 2011.
15. J. Bartovský, P. Dokládal, E. Dokládalová, and M. Bilodeau. Fast streaming algorithm for 1-d morphological opening and closing on 2-d support. In P. Soille, M. Pesaresi, , and G.K. Ouzounis, editors, *ISMM 2011*, volume 6671 of *LNCS*, pages 296–305. Springer, July 2011.
16. nVidia Corporation. NVIDIA GPU Computing Developer Home Page. `http://developer.nvidia.com/category/zone/cuda-zone`, Jun 2011.
17. Khronos Group. OpenCL. `http://www.khronos.org/opencl/`, 2011.
18. B. Obara. Identification of transcrystalline microcracks observed in microscope images of a dolomite structure using image analysis methods based on linear structuring element processing. *Computers and Geosciences*, 33:151–158, 2007.
19. F. Zana and J.-C. Klein. Segmentation of vessel-like patterns using mathematical morphology and curvature evaluation. *IEEE Transactions on Image Processing*, 10:1010–1019, 2001.
20. Willow Garage. OpenCV_GPU. `http://opencv.willowgarage.com/wiki/OpenCV_GPU`, Oct 2011.
21. P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
22. J. Pecht. Speeding-up successive minkowski operations with bit-plane computers. *Pattern Recognition Letters*, 3(2):113–117, 1985.
23. D. Coltuc and I. Pitas. On fast running max-min filtering. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(8):660–663, 1997.
24. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters*, 13(7):517–521, 1992.
25. J. Gil and M. Werman. Computing 2-d min, median, and max filters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(5):504–507, 1993.
26. P. Soille, E.J. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):562–567, 1996.
27. C. Clienti, M. Bilodeau, and S. Beucher. An efficient hardware architecture without line memories for morphological image processing. In *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, ACIVS '08, pages 147–156, Berlin, Heidelberg, 2008. Springer-Verlag.

28. M. Van Droogenbroeck and M.J. Buckley. Morphological erosions and openings: fast algorithms based on anchors. *Journal of Mathematical Imaging and Vision*, 22(2):121–142, 2005.
29. L. Garrido, P. Salembier, and D. Garcia. Extensive operators in partition lattices for image sequence analysis. In *Signal Processing*, pages 157–180, 1998.
30. P. Matas, E. Dokládalová, M. Akil, T. Grandpierre, L. Najman, M. Poupa, and V. Georgiev. Parallel algorithm for concurrent computation of connected component tree. In *Advanced Concepts for Intelligent Vision Systems*, pages 230–241. Springer, 2008.
31. J. Brambor. *Algorithmes de la Morphologie Mathématique pour les architectures orietées flux*. PhD thesis, École des Mines de Paris, 2006.
32. Ch. Clienti. Fulguro image processing library. `http://sourceforge.net/projects/fulguro/`, 2011.
33. L. Domanski, P. Vallotton, and D. Wang. Parallel van Herk/Gil-Werman image morphology on GPUs using CUDA. GTC 2009 Conference posters, 2009.
34. P. Dokládal and E. Dokládalová. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22:411–420, 2011.
35. M. H. F. Wilkinson, H. Gao, W. H. Hesselink, J.-E. Jonker, and A. Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1800–1813, 2008.
36. D. Menotti-Gomes, L. Najman, and A. de Albuquerque Araújo. 1D Component tree in linear time and space and its application to gray-level image multithresholding. In *International Symposium on Mathematical Morphology'07*, volume 1, pages 437–448. INPE, 2007.
37. nVidia Corporation. *CUDA Toolkit Reference Manual*. 2701 San Tomas Expressway, Santa Clara, CA95050, August 2010.
38. J. Kong, M. Dimitrov, Y. Yang, J. Liyanage, L. Cao, J. Staples, M. Mantor, and H. Zhou. Accelerating MATLAB Image Processing Toolbox functions on GPUs. In *GPGPU '10: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 75–85, New York, NY, USA, 2010. ACM.
39. nVidia Corporation. FERMI Tuning Guide. `http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/Fermi_Tuning_Guide.pdf`, Aug 2010.
40. AMD Corporation. OpenCL Course: Introduction to OpenCL Programming. `http://developer.amd.com/zones/OpenCLZone/courses/pages/Introduction-OpenCL-Programming-May-2010.aspx`, May 2010.
41. nVidia Corporation. NVIDIA Parallel Nsight. `http://developer.nvidia.com/nvidia-parallel-nsight`, 2011.
42. J. Nickolls and W.J. Dally. The GPU Computing Era. *IEEE Micro*, 30:56–69, March 2010.
43. nVidia Corporation. CUDA SDK Code Samples. `http://developer.nvidia.com/cuda-toolkit-32-downloads`, Aug 2010.
44. nVidia Corporation. NVIDIA Performance Primitives (NPP) Library User Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/NPP_Library.pdf`, Feb 2011.
45. nVidia Corporation. CUDA C Programming Guide. `http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf`, Sep 2010.
46. T. Randen. Brodatz Textures. `http://www.ux.uis.no/~tranden/brodatz.html`.
47. Pavel Karas and Jan Bartovsky. CUDA-based Linear Openings. `http://sourceforge.net/p/linearopenings/`, Jan 2012.

JRTIP manuscript No.
(will be inserted by the editor)

Jan Bártovský · Petr Dokládal · Eva Dokládalová · Vjaceslav Georgiev

Parallel Implementation of Sequential Morphological Filters

Received: date / Revised: date

Abstract Many useful morphological filters are built as more or less long concatenations of erosions and dilations: openings, closings, size distributions, sequential filters, etc.

An efficient implementation of such concatenation would allow all the sequentially concatenated operators run simultaneously, on the time-delayed data. A recent algorithm (see below) for the morphological dilation/erosion allows such inter-operator parallelism.

This paper introduces an additional, intra-operator level of parallelism in this dilation/erosion algorithm. Realized in a dedicated hardware, for rectangular structuring elements with programmable size, such an implementation allows to obtain previously unachievable, real-time performances for these traditionally costly operators. Low latency and memory requirements are the main benefits when the performance is not deteriorated even for long concatenations or high-resolution images.

Keywords Mathematical Morphology, Serial Filters, Real-Time Implementation, Dedicated Hardware

1 Introduction

Mathematical Morphology is very popular, self-contained, image processing framework providing a complete set of tools from filtering, multi-scale image analysis to pattern recognition. It has been used in a number of applications, including the biomedical and medical imaging, video surveillance, industrial control, video compression, stereology, or remote sensing ([8, 16, 18, 19]).

In image-interpretation applications requiring a high correct-decision liability, one often use robust multi-criteria and/or multi-scale analysis. It generally consists of a serial concatenation of alternating atomic operators dilation and erosion with a progressively increasing computing window, the so-called structuring element (SE). Its examples include:

- *Alternate Sequential Filters (ASF)* - that are concatenations of openings and closings with a progressively increasing structuring element, useful for multi-scale analysis [19, 20].
- *Size distributions* - (aka granulometries) are concatenations of openings allowing measuring the size distribution in a population of objects [14, 17, 28].
- *Statistical learning* - a selected set of morphological operators ζ_i can be separately applied to an image f. Then for every pixel $f(x,y)$, the vector of values $\zeta_i(f)(x,y)$ can serve as a vector of descriptors for the pixel-wise learning and classification [4].

Although built from basic blocks (the dilation and erosion), these operators are costly due to the number of iterations. The real-time capabilities (i.e., low latency) are even more difficult to achieve due to the sequential data dependence and high memory requirements.

The recently introduced algorithm for the dilation and erosion [7] shows how to handle efficiently the implementation of such concatenations. It enables an inter-operator level of parallelism where all the sequentially concatenated operators can run simultaneously, on time-delayed data. Obviously, it is fully exploited only if the algorithm is implemented in an adequate hardware (HW).

In this paper, we propose a HW implementation of the original algorithm and we introduce an additional, intra-operator level of parallelism. Such an implementation allows obtaining previously unachievable, real-time performances for these traditionally costly operators. Section 2

J. Bártovský and E. Dokládalová
Laboratoire Informatique Gaspard Monge
CNRS-UMLV-ESIEE (UMR 8049)
University Paris-Est
93162 Noisy-le-Grand Cedex, France
E-mail: {bartovsj,dokladae}@esiee.fr

P. Dokládal
Center of Mathematical Morphology (CMM)
Mines ParisTech
77305 Fontainebleau Cedex, France
E-mail: petr.dokladal@mines-paristech.fr

V. Georgiev
Faculty of Electrical Engineering
University of West Bohemia
30614, Pilsen, Czech Republic
E-mail: georg@kae.zcu.cz

discusses the state of the art of existing dilation/erosion algorithms and concludes by the novelties presented in this paper. Section 3 and 4 recall the definitions and algorithmic principles and Section 5 illustrates the functional scheme of a sequential HW implementation. Then, Section 6 introduces the parallel implementation allowing obtaining an additional performance increase. Finally, Section 7 presents results obtained on FPGA.

2 Fast Implementations of Morphological Filters

During the last decades, propositions of optimized implementations concentrated on the efficiency of computing the dilation and erosion. The majority of authors measures this efficiency as a number of comparisons per pixel. Nevertheless, the minimization of comparisons can result in high memory requirements. It can even penalize the execution time since the overall latency issues are neglected.

For the following, we define as *operator latency* the latency introduced by the dependence of the result on future data samples. For example the max filter $y_i = \max(x_{i-2}, x_{i-1}, \ldots, x_{i+2})$ has operator latency 2. We define as *algorithm latency* any additional latency introduced by the algorithm, e.g., the necessity to perform a reverse scan on data. The latency is a time-less measure expressed in a number of data samples.

Note that there is also an additional delay, called *computing latency*, induced by the time needed to compute the result after all data are available. It is a platform dependent measure, independent of two previous latency definitions. In the example above, the polyadic max can either be executed sequentially on sequential machines, or in parallel on the dedicated hardware.

Then the overall latency of the system is the sum of these three terms.

2.1 Algorithmic Advances

The most efficient dilation/erosion algorithms are based on the SE decomposition to a set of basic, more easily optimized shapes, see [22, 30, 31]. A special attention is paid to 1-D algorithms obtaining a significant gain in the overall performance.

The most popular 1-D algorithm is called HGW (published by van Herk [26], and Gill and Werman [9]). The computation complexity per pixel is $\mathcal{O}(1)$, i.e., is independent of the SE size. Nonetheless, the algorithm requires two scans: forward and reverse. Lemonnier [12] proposes to identify local extrema and propagate their values as long as it is covered by the SE. Again, forward and reverse scans are needed. Notice that in 2-D the reverse scan of the vertical component multiplies the algorithm latency by a factor of the image width.

Lemire [11] proposes a fast, stream-processing algorithm for causal linear SE. It runs also on floating-point data, has low memory requirements and zero algorithm latency. However, an intermediate storage of local maxima results in a random access to the input data. This problem is solved in Dokladal and Dokladalova [7] using the strictly sequential access to the data. It allows the real on-the-fly computing and has zero algorithm latency.

A different approach represents the algorithm proposed by Buckley and Van Droogenbroeck [25]. It detects the anchors – the portions of the signal unaffected by the operator – and updates only the parts to be modified by the operation. It has zero algorithm latency. However, the algorithm uses a histogram which makes memory requirements dependent on the number of gray levels.

Recently, Urbach and Wilkinson [24] propose an algorithm for arbitrary shaped 2-D flat SEs based on the computation of multiple horizontal linear SEs for every pixel and storing them in a look-up table. The result is then computed by taking the maximum from the intermediate values (stored in the look-up table) corresponding to the shape of the SE. The horizontal linear SE can be computed with one of the above mentioned 1-D algorithms.

2.2 Implementations

In the beginning of the 70's, Klein and Serra [10] propose a texture analyzer for linear and rectangular SE by decomposition based on the delay-line concept. More recently, Velten and Kummert [27] propose also a delay-line based architecture supporting arbitrary-shaped SEs. However, the complexity being quadratic $\mathcal{O}(W^2)$ (W denotes the length of the SE), it becomes penalizing for large SEs. In Chien *et al.* [2], the authors show how to reduce the number of redundant comparisons within large SEs by merging adjacent smaller SEs. The complexity becomes $\mathcal{O}(\lceil \log_2(W) \rceil)$ with identical memory requirements.

Clienti *et al.* [3] proposes a highly parallel morphological System-on-Chip. It is based on a set of neighbourhood processors optimized for 3×3 SE, interconnected in a partially configurable pipeline. Larger SE is obtained by homothecy (see Basic Notions below) requiring to instantiate a deep pipe of these processors or multiple image scans.

A similar approach has been published by Deforges *et al.* [5]. Based on Xu's [30] decomposition combined with a stream implementation, the authors propose a pipeline architecture composed of the elementary parametrizable blocks. It handles an arbitrary convex shape of structuring element in only one scan of the input image. However, using large SE will require the proportional increase of the atomic HW resources, concatenated in a deep pipe. The principal limitation comes from a limited programmability of such a pipe.

To complete this brief survey, we can also cite the systolic architectures proposed by Diamantaras and Kung [6], Malamas *et al.* [13] or Shih *et al.* [21] for gray-scale or binary morphology. Their common inconvenience is the need of an intermediate storage for 2-D structuring element and a long response time of the system.

2.3 Novelty of this Paper

All previous algorithms optimize the dilation/erosion algorithm, rather than the entire operator chain. The performance will inevitably decrease for more complex applications with long loops (iterations, idempotence) or concatenations.

Consider some serial morphological filter $\zeta = \delta\varepsilon \ldots \delta\varepsilon$ (with δ and ε standing for dilation and erosion, see Section 3 below for details). If the atomic operators δ and ε use sequential access to data then the entire ζ can run on pipelined, time-delayed data. If the atomic algorithms δ and ε - in addition - have zero algorithm latency, then the entire chain ζ inherits the same properties: sequential data access and zero algorithm latency. This is an interesting property, since computing ζ suddenly becomes very efficient: in stream, with only the (further irreducible) operator latency of ζ.

In comparison with the preceding state of the art, the Dokladal [7] algorithm extends the possibility to implement erosion/dilation filters with the arbitrarily large, 2-D SE in only one scan over the image, with the minimal algorithm latency and memory requirements. If implemented in a dedicated hardware, we can obtain the same features even for the implementation of long concatenations ζ.

This paper starts from the sequential HW implementation of the Dokladal algorithm (published in Bartovsky *et al.* [1]). It describes more deeply the implementation features and optimization techniques. It shows how to exploit the inter-operator parallelism in ζ. Additionally, it introduces another intra-operator parallelism in the computation of the 2-D erosion/dilation. The 2-D erosion/dilation is implemented as a run-time programmable block. The operation (erosion or dilation), the size and the origin of the SE can be modified on-the-fly between two frames.

Several such blocks concatenated in a pipeline allow obtaining previously unachievable, real-time performances for operators in the form of ζ. We can reach almost 100Hz HDTV 1080p performance, independent of the length of ζ.

3 Basic principles

Let δ_B, ε_B: $\mathbb{Z}^2 \to \mathbb{R}$ be a dilation and an erosion on grey-scale images, parametrized by a structuring element B, assumed rectangular, flat (i.e., $B \subset \mathbb{Z}^2$) and translation-invariant, defined as

$$\delta_B(f) = \bigvee_{b\in B} f_b \tag{1}$$
$$\varepsilon_B(f) = \bigwedge_{b\in \widehat{B}} f_b \tag{2}$$

The hat $\widehat{\ }$ denotes the transposition of the structuring element, equal to the set reflection $\widehat{B} = \{x \mid -x \in B\}$, and f_b denotes the translation of the function f by some vector b. The SE B is equipped with an origin $x \in B$. Below, $B(x)$ denotes B placed with its origin at x.

The implementation of (1) and (2) consists of searching the extremum of f within the scope of B

$$[\delta_B(f)](x) = \max_{b\in B} [f(x-b)] \tag{3}$$

$$[\varepsilon_B(f)](x) = \min_{b\in B} [f(x+b)] \tag{4}$$

The dilation and erosion by convex structuring elements verify the homothecy. Let B be some convex structuring element, and rB the change of scale of B, with $r > 1$, $r \in \mathbb{Z}$. Then for the dilation we have

$$\delta_{rB} f = \underbrace{\delta_B \ldots \delta_B}_{r-\text{times}} f. \tag{5}$$

The homothecy allows obtaining large-size dilations by repeating several times the dilation by a small SE.

Combinations of dilations and erosions form other operators. The basic concatenation products are opening $\gamma_B = \delta_B\varepsilon_B$ and closing $\varphi_B = \varepsilon_B\delta_B$. From here we can form the Alternating Filters obtained as $\gamma\varphi$, $\varphi\gamma$, $\gamma\varphi\gamma$ and $\varphi\gamma\varphi$. The number of combinations obtained from two filters is rather limited. Other filters can be obtained by combining two *families* of filters. This leads to morphological Alternate Sequential Filters (ASF), originally proposed by Sternberg [23], and studied in Serra [19], Chapter 10. In general, it is a family of operators parametrized by some $\lambda \in \mathbb{Z}^+$, obtained by the alternating concatenation of two families of increasing and decreasing filters $\{\xi_i\}$ and $\{\psi_i\}$, respectively, such that $\psi_n \le \ldots \le \psi_1 \le \xi_1 \le \ldots \le \xi_n$.

The most known ASF are those based on openings and closings, obtained by taking $\psi = \gamma$ and $\xi = \varphi$:

$$ASF^\lambda = \gamma^\lambda \varphi^\lambda \ldots \gamma^1 \varphi^1 \tag{6}$$

starting with a closing, and

$$ASF^\lambda = \varphi^\lambda \gamma^\lambda \ldots \varphi^1 \gamma^1 \tag{7}$$

starting with an opening.

The second application example is the size distribution of a population of objects [14, 17, 28]. One way to compute them is the residue from a sequence of openings

$$sd(\lambda) = \|\gamma^\lambda - \gamma^{\lambda-1}\| \tag{8}$$

The following section briefly recalls the principles of the used algorithm, [7]. It starts by the 1-D dilation algorithm, followed by the principle of separation of the n-D dilation into perpendicular 1-D computations, preserving the stream aspects at all levels.

4 Algorithm Description

4.1 1-D Dilation Algorithm

The algorithm principles and properties have been originally published in [7]. We briefly recall the main important principles for HW implementation.

For some 1-D input signal f: $1 \ldots N \to \mathbb{R}$, the algorithm[1] computes the value $\delta_B f(wp) = f(rp)$. The SE B, $B \subset \mathbb{Z}$,

[1] See Appendix for the 1-D dilation pseudocode

is a line segment, containing its origin, and not necessarily symmetric. Consequently, the size of B is given by the span from the centre to the left and to the right, $SE1$ and $SE2$. The length of B is $SE1+SE2+1$. The coordinates wp and rp stand for the current *writing* and *reading* positions.

The algorithm uses a FIFO queue Q. The queue supports operations *push*, *pop* and *dequeue* (modifying the FIFO's content) and queries *front* and *back*. The input signal f is read sequentially. A newly read value $f := f(rp)$ is inserted in the FIFO queue as a pair $\{f, rp\}$, the sample f and reading position rp (code line 3). In this pair, one can independently access either the value or the position by indexing. For example the last stored element's value can be accessed (without dequeing it) by a query Q.back()[1].

The algorithm does not store non decreasing intervals (see [7] for details and proof). The values that appear to belong to increasing or constant intervals are dequeued (code lines 1-2). Consequently, the values stored in the queue are always ordered in a decreasing order.

The old values, uncovered by the SE, are retrieved from the queue (code lines 4-5). The result of the dilation $\delta_B f(x)$ is read at the front of the queue (code line 7). The result becomes available as soon as enough input data have been read, otherwise the output is empty (code line 9).

4.2 2-D Dilation Algorithm

The separability of n-D morphological dilation into lower dimensions is a well known property. For example, a rectangular SE R decomposes as $R = H \oplus V$ where H and V are horizontal and vertical segments and $\oplus$ is the Minkowski addition. Then the dilation by a rectangle R can be computed by concatenation of two perpendicular 1-D dilations

$$\delta_R = \delta_V \delta_H. \tag{9}$$

The sequential access to the data in 1-D makes that two perpendicular 1-D computations can be assembled into 2-D with sequential access at both levels, 2-D and 1-D, for both input and output data. There is no additional latency and no intermediate storage (the data are pipelined).

See the example of dilation by a rectangle $R{=}H \oplus V$ of an $N \times M$ image f, Fig. 1. The image is sequentially read in the raster-scan mode, line by line from left to right. The various indices rp and wp denote *reading* and *writing positions*, respectively, for the segments H and V, and the rectangle R. The computation is illustrated for column i and line j, i.e., the result $\delta_R f(i, j)$ is to be written at wp_R.

The computation of $\delta_R = \delta_V \delta_H$ decomposes as follows: The current reading position of δ_R coincides with rp_H, that is $rp_R = rp_H$. The result of the horizontal dilation, at wp_H, is immediately read by the vertical dilation in the respective column, that is $wp_H = rp_V$. The result of the vertical dilation δ_V is written at the writing position wp_R, i.e., $\delta_V = wp_R$.

Notes:

- The rp_r and wp_r run over the image in the raster scan mode. The distance between rp_R and wp_R is the (further irreducible) operator latency.
- There is one instance of the horizontal dilation running at the current line j, and N instances of vertical dilation, i.e., one per each column.

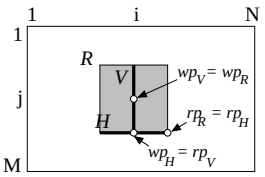


Fig. 1 Decomposition of dilation by a rectangle R into two 1-D dilations by segments H and V, see (9).

5 Sequential Hardware Implementation

In this section, we firstly describe in details the implementation of the 1-D dilation in a basic block that (thanks to the separability) can be used as a building brick in any dimensional system. We illustrate this below on a 2-D dilation or erosion. Secondly, we show how the intra-operator parallelism can be introduced to increase the performance.

5.1 1-D Algorithm Implementation

The 1-D algorithm presented in Section 4 is a system with sequential behavior. It contains a *while* loop that can not be unrolled (uncertain number of iterations). The common way to implement such a system is the Mealy Finite-State Machine (FSM, see [15]). The FSM issues all the necessary operations over the memory as well as it controls the input and output data-flow. It consists of 2 states $\{S1, S2\}$.

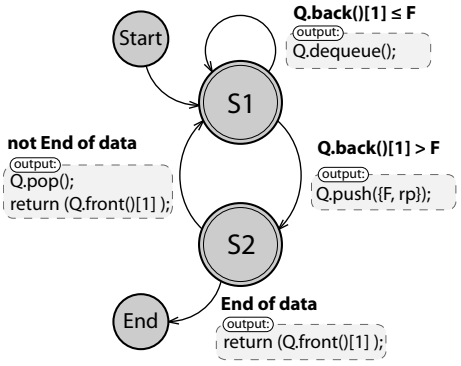


Fig. 2 State diagram of the 1-D Algorithm FSM. State transition conditions are typeset in bold; the output signals are given in a shadow bounding box.

The $S1$ state *Dequeues all useless values*. It is a data dependent stage of the algorithm as it dequeues an a priori unknown amount of pixels. This is represented in the code by

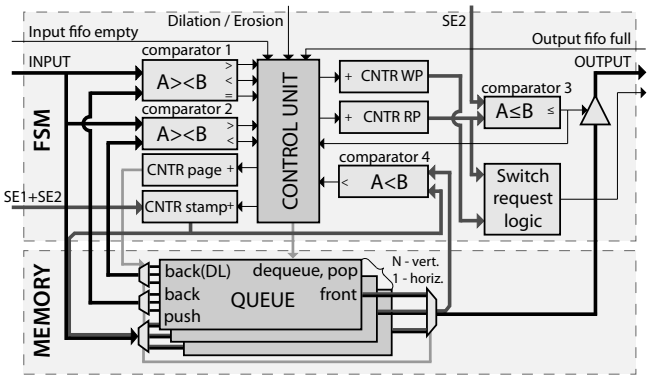


Fig. 3 Overview of implemented 1-D architecture. The FSM part manages computation, memory part contains the data storage–queues

the *while* statement (code line 1). Consequently, its computation time varies from 1 to the SE size clock cycles in the worst case when all the previously stored pixels are unnecessary.

The *Enqueue current sample* signal (code line 3) is issued upon the transition from $S1$ to $S2$.

The $S2$ state handles the code lines 4 and 5, *Delete too old values*, and the lines 6 to 9 *Return valid value* or *Return empty*. These instructions are independent and executed in parallel. Consequently, the execution of $S2$ takes only one clock cycle.

5.2 1-D Block Architecture

The HW implementation can be separated into 2 areas (Fig. 3), the FSM part and the memory part. The FSM manages entire computing procedure and temporarily stores values in the memory part. The memory instantiates one FIFO queue in the case of horizontal direction (horizontal scan) and N FIFO queues in the vertical case (N is the image width). The queues are addressed by a modulo N Page counter (active in the case of vertical direction).

The Control unit is a sequential circuit that manages the state transitions. It increments the rp, wp and manages the Page and position Stamp counter appropriately. The Control unit also performs the queue memory operations and handles the backward full flags used for data-flow control.

Principle

In the beginning of $S1$, the last queued pixel is invoked by Back() operation from the queue and fetched to the Comparator 1. The pixel value is compared with the value of the current sample. Notice that the comparator evaluates all three possible relations ($>$,$<$,$=$) at the time, for both dilation and erosion. The Control unit decides on the basis of comparison results and selected morphological function (dilation or erosion) whether the enqueued pixel is to be dequeued (lines 1-2). Otherwise, the current pixel is extended with the reading position stamp and enqueued (line 3).

The $S2$ invokes the oldest queued pair {pixel, stamp} by Front() operation. The read pixel is a correct result if rp has already reached or exceeded the $SE2$ parameter. This output allowing condition (line 6) is checked by Comparator 3. The deletion of outdated values is performed by comparing the current value of the reading-position stamp with the rp value of the oldest pair. Notice that the deletion has no impact on the output dilation value because Pop() operation (lines 4 and 5) issued by the Control unit has an effect only with the next clock edge.

The switch request logic is used only in the parallelized version of the architecture, see Section 6. It is a simple block containing several comparators which generate a signal with the last output value of each parallel segment. Its purpose is to inform the switch connected to the output that the end of the segment has been reached and the following segment is to be processed.

The entire set of parameters, i.e., SE dimensions and selection of the morphological function, is run-time programmable at the beginning of the line for 1-D, and of the frame for the 2-D implementation, respectively. In addition, no further controller is needed; the internal behavior is driven only by the regular scan order data-flow.

5.2.1 Reducing the impact of data-dependency

Hereafter, we briefly describe two techniques brought to the system for higher throughput and lesser area occupation.

Number of dequeue steps

The data-dependent number of dequeue steps (below denoted by $Steps$) has an unpleasant consequence on the HW design: longer balancing FIFOs (see Fig. 4), lower data throughput. For HW design it is important to minimize the worst case upper bound $Steps_{orig}{=}SE-1$.

The number of stored pixels is within $[1,SE[$. Suppose that we are to dequeue D pixels. We know that the pixels are queued in a strictly decreasing order. Thus, if the DL-th pixel ($DL< D$) can be dequeued then also all previous pixels can be dequeued. This can be done at the same time. Hence, the worst-case number of dequeue steps is

$$Steps = \max_{D<SE}(D\,\mathrm{div}\,DL + D\,\mathrm{mod}\,DL) \tag{10}$$

where D denotes the number of pixels to be dequeued and div and mod the integer division and the remainder operations. D can be regarded as a uniformly distributed, random variable $D \in [1,SE[$. Then we need to find the optimal DL that minimizes $Steps$ (Eq. 10) for all D such as

$$DL_{optim} = \arg\min_{DL<D}\max_{D<SE}(D\,\mathrm{div}\,DL + D\,\mathrm{mod}\,DL) \tag{11}$$

The optimal DL_{optim} brings us the minimal number of dequeue steps $Steps_{optim}$

$$Steps_{optim} = \min_{DL<D}\max_{D<SE}(D\,\mathrm{div}\,DL + D\,\mathrm{mod}\,DL) \tag{12}$$

Table 1 exemplifies, for some *SE* widths, the original and reduced number of dequeue Steps, obtained with optimal *DL*. Notice that more than one optimal *DL* can exist.

The *SE* is user programmable. DL_{optim} also is programmable, though it is useless to make it accessible to the user; it can instead be read from a LUT for every given user-specified SE.

Table 1 Optimal dequeue length *DL*, original and reduced number of dequeue steps for selected SE widths

SE width	3	11	21	31	41
$Steps_{orig}$	2	10	20	30	40
DL_{optim}	2	3, 4	4, 5, 6	5, 6, 7	6, 7
$Steps_{optim}$	2	4	7	9	10

Pixel addressing

The absolute pixel addressing in the queues can be advantageously replaced in the HW by using the modulo addressing. Instead of the absolute reading position rp, we use the relative modulo position $stamp = rp \bmod SE$. The pixels are enqueued by $Q.push(f, stamp)$ (code line 3).

The delete condition of line 4 changes accordingly. Using the modulo addressing, a stored pixel becomes outdated whenever its modulo address equals the current pixels' one ($stamp = Q.front()[2]$).

The advantage of the modulo addressing is a smaller data width. It fits into $\lceil \log_2(SE-1) \rceil$ bits, whereas the absolute addressing requires $\lceil \log_2(N-1) \rceil$ bits. This is mainly advantageous for vertical orientation using *N* queues for a unit.

5.3 2-D Dilation Implementation

Recall that dilation is separable into lower dimensions, Eq. 9. The dilation by a rectangle can be implemented using two 1-D dilation blocks, Fig. 4.

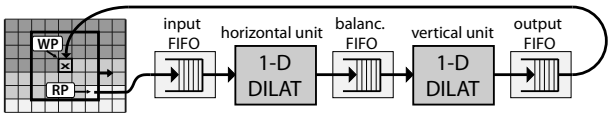


Fig. 4 2-D implementation is composed of 1-D blocks for respective directions.

The computing latency of the dilation varies per each pixel. In order to preserve the input/output stream flow, one needs to compensate the different latencies by insertion of balancing FIFOs. The FIFO fills when the preceding block outputs data faster than the subsequent block can read. The depth of this FIFO directly defines the upper bound of the system latency of the 2-D block.

Obviously, the FIFOs should be as small as possible. The necessary depth infers from the dequeuing worst case

$$F_{input} = \frac{Steps_{hor}+2}{StreamRate} - 1 \tag{13}$$

$$F_{balance} = N\left(\frac{Steps_{ver}+2}{StreamRate} - 1\right) \tag{14}$$

where $Steps_{hor}$ and $Steps_{ver}$ are numbers of the dequeue steps in horizontal and vertical directions (12).

The output FIFO ensures a permanent stream delay in all circumstances. Its maximal size is a sum of both FIFOs (input and balancing). The instantaneous filling of output FIFO is complementary to the filling of both FIFOs combined. The overall delay does not change. If more 2-D blocks are pipelined to form compound operators (e.g., opening, closing, ASF), only one output FIFO at the end is necessary.

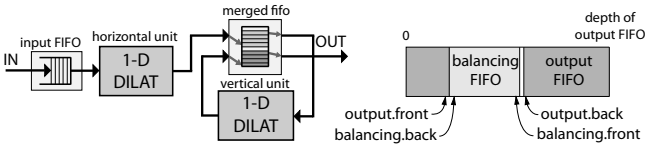


Fig. 5 Merged FIFO replaces the balancing and output FIFOs to reduce memory requirements.

The output and balancing FIFOs can be merged (see Fig. 5) into one memory thanks to the following properties: 1) the vertical unit reads exactly one pixel from the balancing FIFO for each pixel written to the output FIFO. Consequently, filling of these two FIFOs is complementary; the occupied memory spaces can not collide with each other, 2) the read/write activity is at most 1 access per 2 clock cycles. Hence, reading ports of both FIFOs can use one memory port and the writing ports can use the other memory port (without overloading it). Merging both FIFOs reduces the memory to approximately one half. The result memory (see Fig. 5) has two pairs of standard FIFO ports, but it contains only one dual-port RAM.

5.4 Clock rate

The overall average clock rate stays in the interval from 2 clock cycles per pixel in the best case, up to 3 clock cycles per pixel in the worst case. The current rate between 2 and 3 clock cycles per pixel is data dependent.

A temporarily worst case arrives whenever a monotonously decreasing signal is followed by a high value. This makes a number of samples to be dequeued at the time (code lines 1 - 2, and the S1 state of the FSM), and the computing latency temporarily increases. However, the average computing latency remains unchanged, compensated by the fact that during the entire monotonous decrease of the signal no values have been dequeued. Therefore, the average clock cycles per pixel rate remains constant.

5.5 Memory requirements

The memory requirements of the 2-D architecture consist of horizontal and vertical computation-involved memories and two balancing FIFOs, defined by (13) and (14).

In the vertical case, the algorithm uses a several queues. Instantiating N separated memories would be resource inefficient because the FPGA RAM blocks could not be exploited. Instead, these queues are gathered in a single dual-port memory (see Fig. 6) since only one queue is accessed at the time (the others are idle). A single memory block also allows using an off-chip memory.

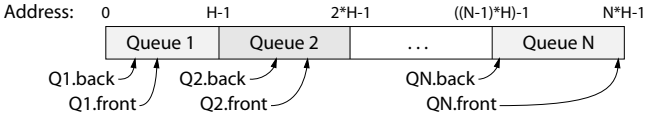


Fig. 6 Vertical Queues are mapped into linear memory space side by side. The front and back pointers are stored at separated memory.

Every queue has a related pair of front and back pointers which must be retained throughout the entire computation process. The appropriate pair is always read before the particular queue is used and the modified pointers are stored back after the computation left the queue. These pointers are stored in a separated pointer memory. The queues are efficiently packed into RAM blocks resulting in a small memory extension.

Let W×H denote the width×height of the rectangular SE, and bpp bits per pixel. The memory contribution per 2-D unit is given by:

$$M_{hor} = W(bpp + \lceil \log_2(W-1) \rceil) \quad \text{[bits]} \quad (15)$$

$$M_{ver} = N(H(bpp + \lceil \log_2(H-1) \rceil) + 2\lceil \log_2(H-1) \rceil) \quad \text{[bits]} \quad (16)$$

The following example illustrates the very low memory consumption achieved thanks to the stream processing. Neither the input, nor the output or any working image are buffered.

Example: Consider a dilation of 8bpp, SVGA image (i.e., 800×600=N×M) by a square, 31x31 SE.

The computation (the queues) requires (15) and (16)

$$M_{hor} = 31(8+5) = 403 \text{ bits}$$

and

$$M_{ver} = 800(31(8+5)+2\times 5) = 330.4 \text{ kbits}$$

resulting in a total of 331 kbits for the 2-D dilation.

The input and the balancing FIFOs require (13) and (14)

$$F_{input} + F_{balance} = (N+1)(\frac{Steps+2}{StreamRate} - 1)8\text{bpp} = \\ = (800+1)(\frac{9+2}{3} - 1)8\text{bpp} \approx 17 \text{ kbits}$$

The total memory needs to implement the 2-D dilation are 331+17=349 kbits. This is far below the mere size of the image itself $800 \times 600 \times 8\text{bpp} = 3.84$ Mbits which, at any moment, does not need to be stored.

6 Parallel Hardware Implementation

This section develops and implements the concept of the previously mentioned *intra-operator parallelism* in the dilation/erosion operator. Its main objective is to increase the throughput while maintaining the beneficial properties of the proposed algorithm, namely the sequential data access and minimal algorithm latency as much as possible.

The principle is based on utilization of concurrently working units that process different parts of the image simultaneously. The number of units used in parallel for horizontal and vertical directions defines the parallelism degree (PD). Considering that the input data are fetched line by line, we propose a solution minimizing the waiting-for-data periods of all units.

The image partition for 2-D dilation conforms to the intersection of two horizontal and vertical partitions (Fig. 7). Its granularity is determined by the PD. The horizontal partition (partition of image among horizontal units) is interleaved, whereas the vertical units use the partition into compact blocks.

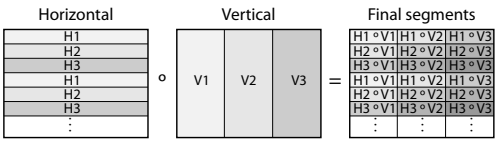


Fig. 7 Example of image partition for *PD*=3: image is divided horizontally line by line and into *PD* equal stripes in a vertical direction. The final image partition is obtained by intersection.

During the parallel processing the computation runs simultaneously at multiple segments of the image, see Fig. 8. These segments must belong to different columns and lines,i.e., must be placed on a diagonal.

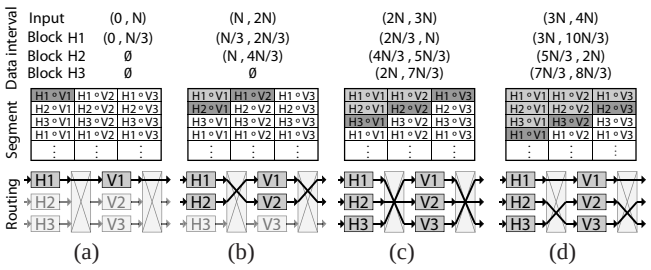


Fig. 8 Image partitioning and switch routing in parallel processing for *PD*=3. Decomposed in time - (a) beginning of processing, (b .. d) after kN pixels, k=1..3. The shading denotes the state : Dark Gray - being computed, Light Gray - already computed, White - waiting.

The input data rate can be theoretically *PD*-times faster than the computational throughput of one unit. Therefore, each image line needs to be buffered in a line buffer. The line buffers are filled at the external (fast) pixel rate and read by the internal *PD*-times slower rate.

Figure 8 gives an example for $PD = 3$. We have three horizontal (H1 .. H3), and tree vertical (V1 .. V3) processing units. As soon as the line buffer receives the first pixel, the first horizontal unit H1 starts the processing and feeds results to the first vertical unit V1. Its output is fed to the first output line, see Fig. 8(a). After *N* received pixels, the output of H1 is connected to V2 which belongs to output line 1. Since the H1 left V1 and line 2 is read, the H2 can start processing second line feeding V1 connected to output line 2, see Fig. 8(b). When the 2N input pixel is received, the H1 connects to V3, H2 connects to V2 and H3 connects to V1, see Fig. 8(c), and so on.

6.1 Architecture

The parallel architecture depicted in Fig. 9 contains four separable generic parts scalable by $n \equiv PD$: input buffer, horizontal and vertical parts and output buffer. The input buffer is mainly composed of the 1-to-*n* multiplexer and *n* line buffers (we omitted the control logic). It divides the fast input stream into *n* (*n*-times slower) streams processed by computational units as described above. The output buffer composes *n* slow streams of the processed data into a single, fast, output stream respecting the image horizontal scan order. The operator blocks can be concatenated into more complex functions (opening, closing, ASF, etc.). The buffers are used only at the beginning and at the end of the chain.

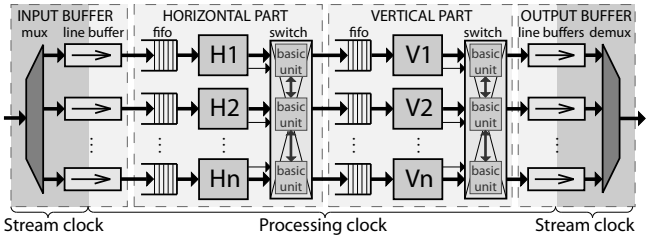


Fig. 9 Overview parallel 2-D architecture. The horizontal and vertical stages can be instantiated several times between input/output buffers to create compound operators.

Both horizontal and vertical parts instantiate *n* balancing FIFOs, *n* horizontal or vertical units, and one switch that manages the interconnection. Each horizontal unit along with the front-end FIFO conforms to Section 5.2.

The width of the processing area proportionally affects both vertical memories, see (14) and (16). The area of every horizontal unit remains unchanged, since every unit processes the entire line. The overall memory of the horizontal part is a factor of *n*. Contrarily, the memory requirements of every vertical part is divided by *n* because it processes only a fraction of the original image width. The area of the FSM of vertical units increases linearly with *n*.

6.2 Switching

The routing of the computation units is handled by a switch block. Every switch contains *n* input ports from previous units and the same number of output ports linked to the subsequent units. The purpose of the switch is to manage up to *n* interconnection channels. Notice that they are bidirectional: forward data and backward FIFO full flag. As described in Fig. 8, the output switching of all input ports is circular, i.e., V1, V2 ... Vn, V1, V2, ... and so forth. This property makes the switching easier because the only condition to evaluate is when to switch and whether the requested output unit is available.

The moment when to switch a given port is provided by the preceding unit's *Switch Request* logic. It generates a request every time it crosses the border of adjacent segments. If the desired unit is free, the switch reconnects the channel. If not, the switch sets high the FIFO full flag of requesting unit to stall it until the desired destination unit is freed and the channel can be established. All the channels are switched independently so stalling one unit does not affect the others.

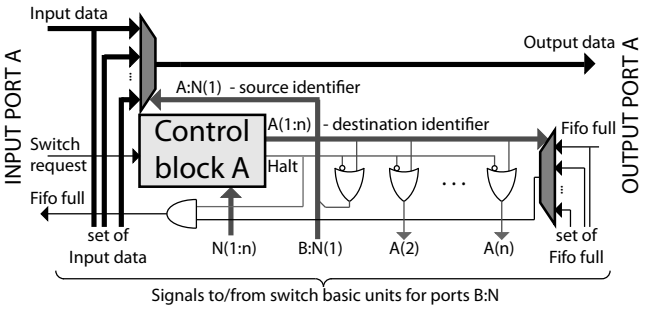


Fig. 10 Basic unit of the switch. Every switch contains *n* basic units for a correct routing between *n* input/output ports.

Figure 10 depicts the basic unit of the switch for one pair of input/output ports referred to as A. For *n* pairs of ports this circuitry is instantiated *n*-times. Each input port possesses a related control unit block that manages all channel transitions considering the availability of the requested partition. If this is still occupied, the requesting computation unit is stalled by holding its FIFO full flag active.

7 Experimental results

The proposed 2-D stream processing architectures have been implemented in VHDL, and targeted to the Xilinx Virtex5 FPGA (XC5VSX95T-2) using the XST synthesis tool. The processing clock frequency is 100 MHz. Notice that the queues are gathered in a block RAM memory, and thus its access time augments the critical path delay.

The measured performance for non-parallel architectures (*PD*=1) in terms of overall latency, clock cycles per pixel and FPGA area are given by Tables 2 and 3.

Table 2 Timing and area vs. SE, SVGA image size, *PD*=1.

Size of SE (sq.)	3x3	11x11	21x21	31x31	41x41
Latency [clk]	1908	9474	18888	28351	37969
Av. rate [clk/px]	2.344	2.356	2.360	2.361	2.361
Registers	212	232	242	242	252
LUTs	584	761	859	859	953
Block RAMs	2	6	13	13	28

Table 3 Timing, frame rate and area w.r.t. image, SE = 31x31 square, *PD*=1.

Size of Image	CIF	VGA	SVGA	XGA	1080p
Latency [clk]	12826	23465	28351	37472	69548
Av. rate [clk/px]	2.371	2.376	2.361	2.383	2.368
Experimental FPS	384	130	85	51	20.5
Worst-case FPS	319	106	68	41	16
Registers	231	237	242	242	253
LUTs	761	853	859	859	1057
Block RAMs	7	13	13	13	26

One can observe that the overall latency is factor of the SE size, the image width (both caused by operator latency) and the pixel rate (computing latency). The average pixel rate (AR) remains constant (Table 2). The average pixel rate can be expressed by (17) and the stream frame-per-second (FPS) ratio by (18). T_{proc} is overall time consumed by processing and f_{clk}= 100 MHz is clock frequency of computation units.

$$AR = \frac{T_{proc} - 2SE_2(N+M+SE_2)/PD}{NM} \quad \text{[clk/px]} \quad (17)$$

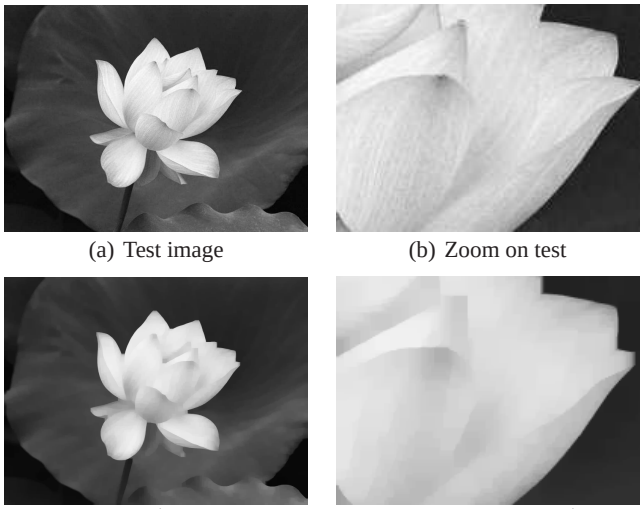
$$FPS = \frac{f_{clk}PD}{ARNM + 2SE_2(N+M+SE_2)} \quad \text{[fr/s]} \quad (18)$$

M, N denote the width and height of the image, SE_2 denotes the width of the structuring element from the origin rightwards.

Concerning the area occupation (see the Xilinx documentation [29]), the number of registers is quasi-constant; the number of LUTs and BRAM blocks increases linearly with the SE and image sizes (Table 3). Although the vertical memory (size is given by (16)) is packed into the RAM block, the amount of the used memory always exceeds the theoretical value. It is caused by a different memory organizations; e.g., the required word is 13 bits whereas available memories are of width 36 bits and its fractions.

The experimental frames-per-second (FPS) rate is obtained on a natural test image (see Fig. 11). The worst-case FPS is a theoretical worst-case performance of the system expected on the synthetic saw-shaped data.

Table 4 presents the relative speed-up of the parallel architecture vs. the intra-operator parallelism *PD*. In terms of overall latency and average processing rate, the processing domain clock cycle is considered as a reference unit. Note that the latencies of parallel versions are merely fractions (divided by *PD*) of non-parallel values.



(a) Test image (b) Zoom on test

(c) ASF^4 filtered (d) Zoom on ASF^4

Fig. 11 (a) Experimental 800×600 lotus image. (b) Zoom on the fine veinous texture disadvantageous for the algorithm. (c) Result of ASF^4 filter, see Eq. 6 or 7. (d) Zoom on the ASF^4 filtered image.

Table 4 Timing vs. degree of intra-operator parallelism *PD*. SVGA image, SE = 31x31 square.

PD	2	3	4	5	6
Latency [clk]	14243	9561	7244	5818	4893
Av. rate [clk/px]	1.220	0.824	0.625	0.505	0.426
Exp. speed up	1.938	2.869	3.785	4.682	5.554

Table 5 Area vs. degree of intra-operator parallelism *PD*. SVGA image, SE = 31x31 square.

PD	2	3	4	5	6
Registers	650	978	1280	1605	1938
LUTs	2138	3227	3862	4875	6054
Block RAMs	13	14	14	18	21
Reg. buf	661	969	1279	1587	1896
LUTs buf	1408	2086	2776	3459	4135

The FPGA area results, Table 5, are separated into 2 groups: the area of computing parts and buffers. The area of input and output buffers is linear w.r.t. both *N* and *PD* since their essential components are *PD* line buffers (FIFO memories with independent ports of *N* elements). The area of the operator units in terms of Slice registers and LUTs is proportional to *PD* as well because *n* independent circuits are instantiated in a parallel manner. Although the overall vertical memory requirements remain unaffected by *PD*, practically the number of occupied RAM blocks slightly increases. It is caused by a different memory organization.

Table 6 Timing and frame rate vs. image size, PD = 6, SE = 31x31 square

Size of Image	CIF	VGA	SVGA	XGA	SXGA	1080p
Latency [clk]	2208	3996	4893	6390	7391	11641
Av. rate [clk/px]	0.443	0.431	0.426	0.426	0.427	0.418
Experimental FPS	2075	724	472	290	174	113
Worst-Case FPS	1915	640	411	246	151	96

The ultimate timing results (*PD*=6) versus the image size are listed in Table 6. It illustrates the real performance of the architecture. It allows to achieve at least 96 fps with 1080p image size (full HD TV image size).

The worst case occurs on artificial saw-shaped image with no constant plateaus. Such an image infers the maximal number of algorithm's while-loop iterations. The best case fps (not mentioned in the table) is obtained with a constant image. A real, unfiltered image containing textures or random noise achieves performance somewhere between best and worst cases. For instance at 1080p, the worst case is 96 fps, best case 140 fps, achieved experimental performance is 113 fps.

This frame rate remains constant for any morphological serial filter (such as ASF). Obviously, the FPGA area increases accordingly to the size of the ASF. The implementation is eased by the fact that one can use an off-chip memory.

7.1 Comparison with existing HW implementations

Table 7 presents a comparison with other recent architectures. The table is divided into three sections. The processing unit section presents the features of a single 2D computational unit. The second part the HW specifications, and the third part the performance on a given application, an ASF filter.

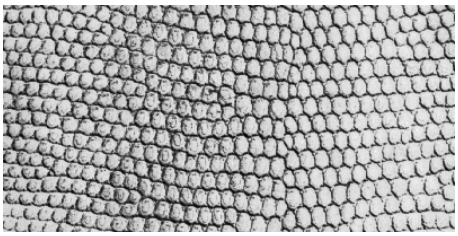
One can see that Clienti [3] offers a high throughput for small 3×3 rectangular SEs. Similarly, the Chien ASIC chip [2] provides very reasonable performance on small SEs. On the other hand, Déforges [5] directly offers large, non-rectangular, convex SE, but with a lower processing rate. The programmability is not mentioned, namely, the possibility to control the SE shape after the synthesis is not clear.

Although all these solutions are efficient for small SE sizes or short concatenations, they become more or less penalized for longer filters. This issue is illustrated in an Example Application, Table 7. It estimates the performances on a five-stage $\text{ASF}^5 = \varphi_{11\times11}\gamma_{11\times11} \ldots \varphi_{3\times3}\gamma_{3\times3}$. Decomposed into a sequence of dilations and erosions, it can be realized as $\text{ASF}^5 = \varepsilon_{11\times11}\delta_{21\times21} \ldots \varepsilon_{5\times5}\delta_{3\times3}$. Notice that it makes use of a progressively increasing SE. On neighborhood processors, large SE can be obtained using the homothecy Eq. 5. The Clienti SPOC instantiates 16 of 3×3 processing units. Hence, the ASF^5 will require 5 image scans with the entire image necessarily buffered in the memory. Chien also uses the homothecy. This deteriorates the throughput.

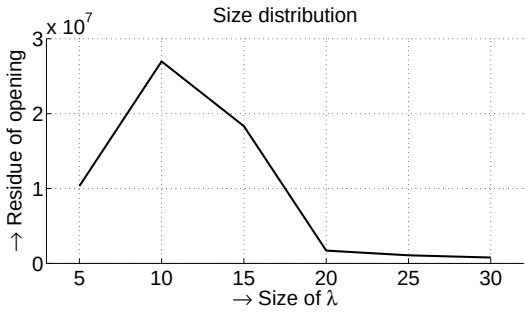
One could immediately figure out to instantiate a longer pipe in order to reduce the number of image scans. Alas, a long, fixed-length pipe lacks the flexibility. Consider another application for the illustration of the problem: the size distributions, exemplified by Fig. 12. Contrarily to ASF, the size distributions are often sampled sparsely, the SE increments by more than one and, at the same time, one often goes to much larger SE sizes. Every opening $\{\gamma_{B_i}\}$ in (8) needs to be output and stored in the memory to compute the subtraction. For small sizes, a long pipeline is underused and the workload of the processing units unbalanced, whereas for large λ one may still need several image scans.

For example, for sizes $\lambda = 5, 10, 15, 20, 25$, as in Fig. 12, the Clienti SPOC will require 7 image scans. The 16 processor pipe is underused for $\lambda = 5, 10, 15$, whereas it will require 2 scans for $\lambda = 20, 25$.

Our processing unit with programmable SE size avoids using the homothecy. This allows optimal workload distribution over the entire pipe, so important for processing large images in real-time systems.



(a) Example of a texture



(b) Size distribution sd(λ)

Fig. 12 The size distribution of the texture grain.

8 Conclusions

This paper describes an efficient implementation of serial morphological filters with flat, rectangular structuring elements of arbitrary size. The efficiency is obtained through the following properties:

− The computational complexity is linear w.r.t. the image size and independent of the SE size.

− The overall latency is mostly equal to the latency of the operator, inferred by the size of the used structuring element.

Table 7 Comparison of several FPGA and ASIC architectures concerning morphological dilation and erosion

	Processing unit					HW System		Example Application ASF[5]	
	Parallel degree	Supported SE	Throughput [Mpx/s]	f_{max} [MHz]	Clock rate [clk/px]	Number of units	Supported image	Image scans	FPS
Clienti [3]	4	arb. 3x3	403	100	0.25	16*	1024x1024	5	80
Chien [2]	1	disk 5x5	190	200	1.052	1	720x480	27	21.5
Déforges [5]	1	arb. convex	50	50	1	1*	512x512	11	17.2
This paper	6	rectangles	234	100	0.426	11*	1920x1080	1	113

* Number of available stages varies with size of used FPGA

– It uses strictly sequential access to the data at all algorithm levels.
– Low memory consumptions (far below the size of the image) allow embedding on a single chip complex operators able to process large images.
– Two levels of parallelism: i) the inter-operator parallelism in serial concatenations $\zeta = \delta\varepsilon \ldots \delta\varepsilon$, allow running all these atomic δ and ε operators simultaneously, and ii) the intra-operator parallelism in every atomic dilation/erosion. The intra-operator parallelism is scalable (tested up to six) and allows the decomposition of fast streams into several slower streams processed in parallel without altering the streaming property of the system.

The architecture serves as a basic building block to be used for construction of more complex operators such as ASF, granulometries, etc., with the same properties and performance. The performances obtained on an FPGA are approaching the 100Hz HDTV 1080p standard. These performances are far above what has been reported in the literature. These performances allied to the programmability are extremely interesting. They open the accessibility of advanced morphological operators in industrial systems running under severe time constraints. The number of examples includes the on-line production control, aging material defectoscopy, etc., wherever one requires processing of high resolution images and low latency.

Appendix: The 1-D Dilation Pseudocode

Algorithm 1: df←1D_DILATION (rp, wp, f, SE1, SE2, N)

Input: rp, wp - reading/writing position; f - input signal value $f(rp)$; SE1, SE2 - SE size towards left and right; N - length of the signal; Q - a FIFO-like queue
Result: output signal value $\delta_B f(wp)$

```
1 while Q.back()[1] ≤ f do
2 |  Q.dequeue() ;                    // Dequeue useless valuess
3 Q.push({f, rp}) ;                   // Enqueue the current sample
4 if wp - SE1 > Q.front()[2] then
5 |  Q.pop() ;                        // Delete too old value
6 if rp = min (N, wp + SE2) then
7 |  return (Q.front()[1] ) ;         // Return valid value
8 else
9 |  return ({}) ;                    // Return empty
```

References

1. J. Bartovský, E. Dokládalová, Petr Dokládal, and V. Georgiev. Pipeline architecture for compound morphological operators. In *ICIP10*, 2010.
2. S.-Y. Chien, S.-Y. Ma, and L.-G. Chen. Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1156 – 1169, sept. 2005.
3. Ch. Clienti, S. Beucher, and M. Bilodeau. A system on chip dedicated to pipeline neighborhood processing for mathematical morphology. In EURASIP, editor, *EUSIPCO 2008*, Lausanne, August 2008.
4. A. Cord, D. Jeulin, and F. Bach. Segmentation of random textures by morphological and linear operators. In *8th ISMM*, pages 387–398, Oct. 2007.
5. O. Déforges, N. Normand, and M. Babel. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. *Journal of Real-Time Image Processing*, pages 1–10, 2010. 10.1007/s11554-010-0171-8.
6. K. I. Diamantaras and S. Y. Kung. A linear systolic array for real-time morphological image processing. *J. VLSI Signal Process. Syst.*, 17(1):43–55, 1997.
7. P. Dokládal and E. Dokládalová. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22(5):411–420, 2011.
8. E.R. Dougherty. *Mathematical morphology in image processing* . Taylor and Francis, Inc., 1992.
9. J. Gil and M. Werman. Computing 2-d min, median, and max filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):504–507, 1993.

10. J.-C. Klein and J. Serra. The texture analyser. *J. of Microscopy*, 95:349–356, 1972.
11. D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *CoRR*, abs/cs/0610046, 2006.
12. F. Lemonnier and J.-C. Klein. Fast dilation by large 1D structuring elements. In *Proc. Int. Workshop Nonlinear Signal and Img. Proc.*, pages 479–482, Greece, Jun. 1995.
13. E. N. Malamas, A. G. Malamos, and T. A. Varvarigou. Fast implementation of binary morphological operations on hardware-efficient systolic architectures. *J. VLSI Signal Process. Syst.*, 25(1):79–93, 2000.
14. P. Maragos. Pattern spectrum and multiscale shape representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):701–716, 1989.
15. G. H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34:1045–1079, 1955.
16. L. Najman and H. Talbot, editors. *Mathematical Morphology: From Theory to Applications*. ISTE Ltd and John Wiley & Sons Inc, 2010.
17. R. Sabourin, G. Genest, and F. Prêteux. Off-line signature verification by local granulometric size distributions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(9):976–988, 1997.
18. J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.
19. J. Serra. *Image Analysis and Mathematical Morphology*, volume 2. Academic Press, NY, 1988.
20. J. Serra and L. Vincent. An overview of morphological filtering. *Circuits Syst. Signal Process.*, 11(1):47–108, 1992.
21. F. Y. Shih, T. K. Chung, and C. C. Pu. Pipeline architectures for recursive morphological operations. *IEEE Trans. Image Processing*, 4(1):11 –18, jan. 1995.
22. P. Soille, E. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(5):562–567, 1996.
23. S. Sternberg. Grayscale morphology. *Comput. Vision Graph. Image Process.*, 35(3):333–355, 1986.
24. E. R. Urbach and M. H. F. Wilkinson. Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements. *IEEE Trans. Image Processing*, 17(1):1 –8, jan. 2008.
25. M. Van Droogenbroeck and M. J. Buckley. Morphological erosions and openings: Fast algorithms based on anchors. *J. Math. Imaging Vis.*, 22(2-3):121–142, 2005.
26. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recogn. Lett.*, 13(7):517–521, 1992.
27. J. Velten and A. Kummert. Implementation of a high-performance hardware architecture for binary morphological image processing operations. In *Circuits and Systems, 2004. MWSCAS '04. The 2004 47th Midwest Symposium on*, volume 2, pages II–241 – II–244 vol.2, 25-28 2004.
28. L. Vincent. Granulometries and opening trees. *Fundamenta Informaticae*, 41(1-2):57–90, January 2000.
29. Xilinx. Virtex-5 family documentation, 2009, available at http://www.xilinx.com/support/documentation/virtex-5.htm.
30. J. Xu. Decomposition of convex polygonal morphological structuring elements into neighborhood subsets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(2):153–162, 1991.
31. X. Zhuang and R. M. Haralick. Morphological structuring element decomposition. *Computer Vision, Graphics, and Image Processing*, 35(3):370–382, 1986.

P^{2}IP: A novel low-latency Programmable Pipeline Image Processor ☆

Paulo Possa[a,*], Naim Harb[a], Eva Dokládalová[b], Carlos Valderrama[a]

[a] *Department of Electronics and Microelectronics, University of Mons Boulevard Dolez 31, 7000 Mons, Belgium*
[b] *Computer Science Laboratory Gaspard-Monge, ESIEE Paris, University Paris-Est 93162 Noisy-le-Grand Cedex, France*

Abstract

This paper presents a novel systolic Coarse-Grained Reconfigurable Architecture for real-time image and video processing called P^{2}IP. The P^{2}IP is a scalable architecture that combines the low-latency characteristic of systolic array architectures with a runtime reconfigurable datapath. Reconfigurability of the P^{2}IP enables it to perform a wide range of image pre-processing tasks directly on a pixel stream. The versatility of the P^{2}IP is demonstrated through three image processing algorithms mapped onto the architecture, implemented in an FPGA-based platform. The obtained results show that the P^{2}IP can achieve up to 129 fps in Full HD 1080p and 32 fps in 4K 2160p what makes it suitable for modern high-definition applications.

Keywords: Reconfigurable hardware, Image processing, Real-time system, Computer vision

1. Introduction

Digital image processing is a well-known class of computationally intensive tasks that conventional computing architectures cannot efficiently

☆This work is supported by the French Community of Belgium under the Research Action ARC-OLIMP (Optimization for Live Interactive Multimedia Processing 2008-2013)

*Corresponding author

Email addresses: paulo.possa@umons.ac.be (Paulo Possa), naim.harb@umons.ac.be (Naim Harb), e.dokladalova@esiee.fr (Eva Dokládalová), carlos.valderrama@umons.ac.be (Carlos Valderrama)

perform in terms of power consumption and when real-time performance is required [1, 2, 3, 4]. Real-time requirements are especially important in latency-critical applications, such as live video broadcasting, video surveillance, Unmanned Vehicle (UV) navigation, interactive multimedia applications, and video assisted medical devices. Latency is defined here as the amount of time between when a signal is impressed on the input of a circuit and when it is received or detected at its output [5].

We can divide image processing tasks into two groups: high-level and low-level tasks [6]. High-level image processing tasks are algorithms that create symbolic representations of the image contents, e.g., object recognition and tracking. Low-level image processing tasks are algorithms that can modify specific aspects in the image (e.g., color correction, filtering, and blurring), or detect features (e.g., edge and corner detectors). The data input of the last group is frequently a stream of a large collection of small and independent data elements [7]. Due to this inherent data parallelism, low-level tasks are very suitable for highly parallel processing architectures. Numerous implementations have been proposed in the past for this last group. They mainly target general flexibility improvement but the problem of minimal latency has not been directly studied.

In order to explore the data parallelism of low-level image processing tasks, we are proposing a new reconfigurable architecture, the Programmable Pipeline Image Processor (P^2IP). The P^2IP is a Coarse-Grained Reconfigurable Architecture (CGRA) based on a linear systolic array model targeting low-latency real-time applications and supporting a wide range of image resolutions and operators. Figure 1 shows a simplified functional diagram of the P^2IP architecture. Each Processing Element (PE) of the P^2IP contains an optimized set of essential image processing operators that can be parametrized at runtime. A Recongurable Interconnection module associated to each PE enables dynamic datapath re-routing. Configuration registers define the behavior of the operators distributed in the P^2IP including the Reconfigurable Interconnection. These registers can be accessed through a dedicated control path in order to change the application context without resynthesizing the system. Algorithms can be created or invoked by using a MATLAB embedded library that supports algorithm allocation and task mapping into the reconfigurable architecture through the configuration port of the P^2IP controller.

The parametrizable PEs and the reconfigurable datapath provide *post-synthesis* flexibility to the P^2IP architecture. Synthesis is defined here as

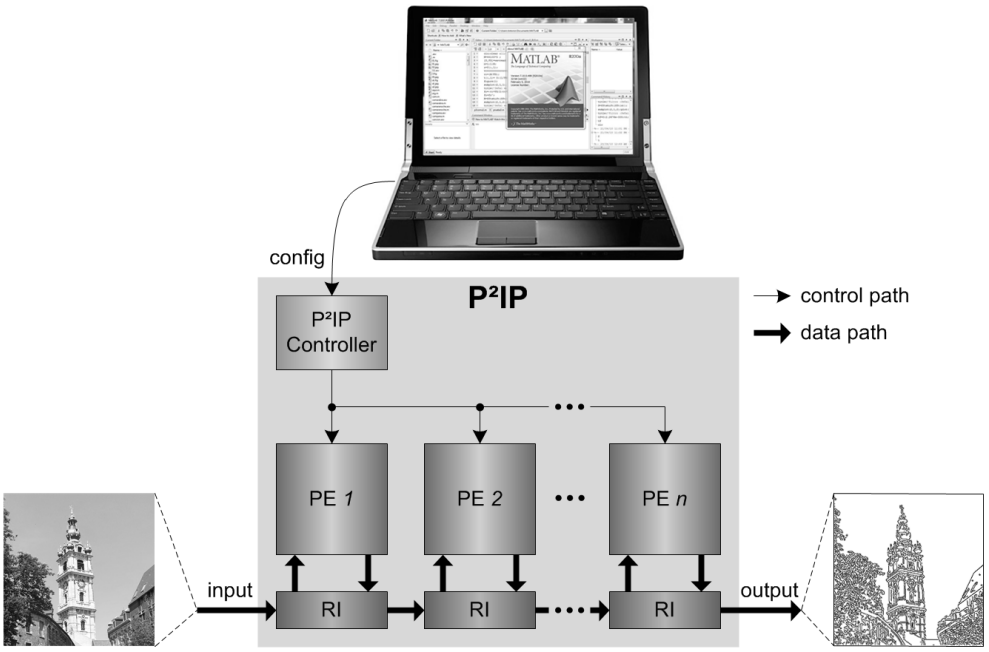


Figure 1: Simplified functional diagram of P^{2}IP architecture showing the datapath formed by Processing Elements (PEs) associated to a Reconfigurable Interconnection (RI).

the process where a digital circuit behavior, usually described in a Hardware Description Language (HDL) such as Verilog HDL or VHDL, is converted by a *synthesizer* into an association of logic gates that can be implemented in hardware. Thus, post-synthesis flexibility means that the P^{2}IP can still be configured to change its functionality after its hardware implementation. Beyond that, the modular organization of P^{2}IP facilitates pre-synthesis customizations in order to meet application requirements. That includes customizations such as the number of PEs, maximum frame resolution, and set of operators.

The rest of this paper is organized as follows: Section 2 presents related works. Section 3 describes the P^{2}IP architecture in detail. Examples of algorithms mapping are presented in Section 4. Afterwards, the algorithm performances and the architecture implementation results are analyzed in Section 5. Finally, the conclusions of this work are presented in Section 6.

2. Related work

It has been almost half a century since one of the first digital computers specially designed for image processing started operating [8]. Since then, a large number of specialized architectures have been designed with the same basic purpose: to process images automatically and efficiently. Historically, the performance requirements of image processing applications have only been met with special-purpose (custom), fixed-function hardware [3, 9], i.e. Application-Specific Integrated Circuits (ASICs) [10]. However, due to the lack of flexibility in custom solutions, alternatives have been proposed by the community in order to provide more flexible architectures. Among these alternatives, three computing models can be highlighted: Domain-specific processors, e.g., Digital Signal Processors (DSPs) [11, 12], Single-Instruction Multiple-Data (SIMD) architectures [1, 13, 14], and Reconfigurable Computing (RC) systems [10, 15, 16].

Among these alternatives, the most promising in terms of computing performance and energy efficiency is the RC approach [15, 17, 18]. RC systems can be divided into two groups, Fine-Grained Reconfigurable Architectures (FGRAs) and CGRAs. FGRAs, e.g., Field Programmable Gate Arrays (FPGAs), are extremely flexible architectures containing PEs in which it is possible to map any 1-bit logic function at the cost of long design and compilation times. CGRAs overcome this cost by using more elaborated PEs that can perform word-level operations for a wide range of applications including image and video processing. Some important examples of these architectures are PipeRench [19], RAW [20], and MORPHEUS [17]. PipeRench is a linear array accelerator for pipelined applications composed of 256 ALU-based PEs that can be reconfigured dynamically. RAW is a multiprocessor architecture containing 16 MIPS-style microprocessors arranged in a 4×4 array interconnected by a both static and dynamic network. MORPHEUS is a more recent architecture including three heterogeneous processing engines, each one with a different reconfigurable granularity, interconnected by a Network-on-Chip (NoC) and controlled by a RISC processor. However, these solutions generate supplementary latency, penalizing the execution time with respect to fixed-function hardware.

In our approach, we combine the flexibility of RC systems to the processing performance and latency of fixed-function hardware solutions to bridge their performance/flexibility gap. The P^2IP flexibility was obtained by exploring classic low-level image processing algorithms, such as Canny Edge

Detection and Harris Corner Detection, resulting in the design of fundamental building operators that could be generalized and combined in PEs to support a wide range of algorithms. The cost of the P^{2}IP approach is to be restricted to a single application domain which in this work corresponds to image and video pre-processing. However, this characteristic reduces algorithm mapping complexity since many image processing operators are already implemented on the architecture, e.g., linear filters and non-maximum suppressors.

Other architectures that offer competitive performances targeting low-power image processing applications are CSX700 [21], Diet SODA [14], and CRISP [22]. CSX700 presents a low-power SIMD architecture for image processing, containing 192 PEs divided into two processing cores. Diet SODA is another low-power architecture targeting Digital Still Cameras (DSCs), containing a 128-lane SIMD unit that works at low frequencies. CRISP processor is a CGRA, but optimized for image and video processing in DSCs. It presents a series of specific image processing operators associated to a configurable interconnection that routes the data stream through these operators, which makes it possible to change the processing task.

It is also interesting to cite here another popular approach for image processing based on General-Purpose Graphics Processing Units (GPGPUs). Though these solutions do not target low power consumption application, they have presented very competitive performances [1, 13]. Also, the work in [23] explores a parallel hybrid approach using optimized heterogeneous multi-CPU/multi-GPU architectures to address image processing tasks.

3. P^{2}IP architecture

The P^{2}IP architecture can be classified as a CGRA based on a linear systolic array model. Images or frames are entered as a stream of pixels in sequential line-scanned format progressing through the pipeline at a constant rate. The P^{2}IP datapath works at the pixel clock frequency and can deliver one processed pixel per clock cycle after the initial latency to fill the pipeline. It was designed to work between a frame source and a frame sink directly on the pixel stream, as shown in Fig. 2.

In order to permit the P^{2}IP integration into an image processing chain, the AXI4-Stream [24] was adopted as the external interconnection protocol. The AXI4-Stream protocol is controlled by the P^{2}IP Controller which is also in charge of reading configuration words on the configuration input

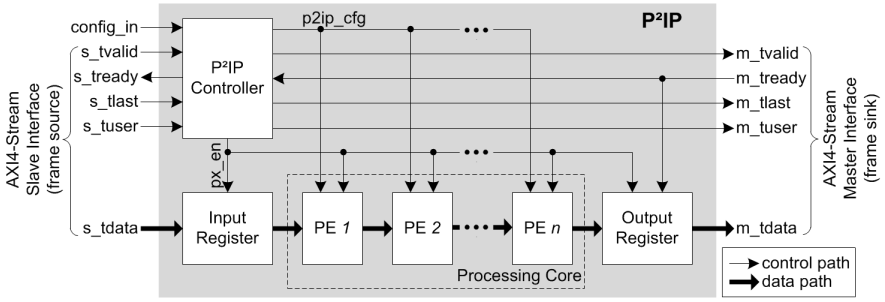


Figure 2: Functional diagram of the P^{2}IP architecture.

port (*config_in*) and their transfer to the PEs. The P^{2}IP Controller is the input node of the P^{2}IP configuration mechanism formed by the *Configuration Tree*. A detailed description of the P^{2}IP configuration mechanism is given in Section 3.2.

The input register block along with the output register block (Fig. 2) constitute the boundaries of the P^{2}IP architecture. This practice prevents any timing critical changes due to nets crossing the module boundaries. Also, it helps synthesis tools to optimize a module without the interference of other system components [25, 26]. The input register block separates three color channels (red, green, and blue) from a true-color input data stream (24 bits per pixel) before transferring it to the first PE in the pipeline. It also generates a fourth color channel (8 bits per pixel) carrying a grayscale version of the input pixel stream. A grayscale image represents an effectively continuous range of tones, from black to white, through intermediate shades of gray [27].

The processing core of the P^{2}IP is formed by a sequence of identical PEs. Although the PEs have the same internal structure, each one can perform a different function according to its configuration. The number of PEs can be defined before synthesis by a simple instantiation procedure without changing the P^{2}IP basic structure or the programming mechanism. Following, the P^{2}IP PE is described in detail.

3.1. The P^{2}IP Processing Element

The PEs are the main components of the P^{2}IP. All PEs are identical, formed by a Reconfigurable Interconnection and modules containing groups of image processing operators. The Reconfigurable Interconnection directs

the pixel stream through modules forming the P^{2}IP datapath. Fig. 3 shows the PE functional diagram.

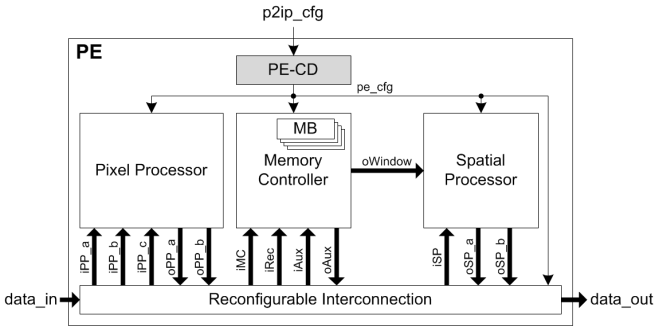


Figure 3: Functional diagram of the P^{2}IP Processing Element (PE).

Each PE has its own local data memory divided into 4 Memory Blocks (MBs) used for storing frame lines (one line per MB). The maximum horizontal size of the input frame is limited by the MB's size which can be defined before synthesis. The MBs are based on dual-port memory elements accessed sequentially. The *Memory Controller* performs all memory operations needed by the PE, e.g., neighborhood extraction and delay. The other modules, *Pixel Processor* and *Spatial Processor*, comprise clusters of image operators. The Pixel Processor module contains pixel-to-pixel operators which process each pixel individually, e.g., arithmetic and logic operations. The Spatial Processor module contains spatial operators which are neighborhood oriented functions. These functions require an input window formed by the targeted pixel and its neighborhood to provide a result. Those three modules are interconnected by the Reconfigurable Interconnection. Also in Fig. 3, we can see the PE-Configuration Decoder (PE-CD) which is the second node of the Configuration Tree. The PE-CD distributes the configuration received from the P^{2}IP Controller to the internal PE modules. More details about this block and the P^{2}IP configuration mechanism are given in Section 3.2. Following, we present a detailed description of the PE internal modules.

3.1.1. Memory Controller

As mentioned earlier, the Memory Controller contains memory-based operators, more specifically, a Neighborhood Extractor (NE), a mirror, and a delay. Fig. 4 shows the functional diagram of the Memory Controller structure.

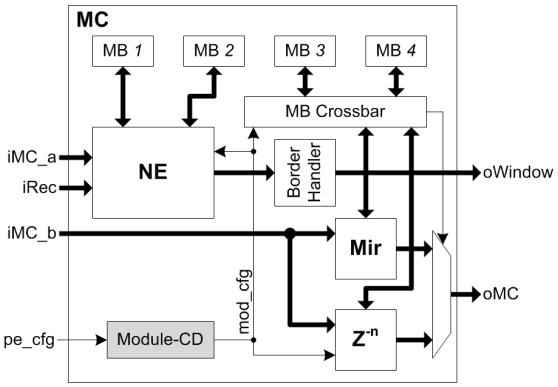


Figure 4: Functional diagram of the Memory Controller (MC).

In order to reduce memory resources, MB 3 and 4 (Fig. 4) can be allocated to any Memory Controller operator through the MB crossbar. It is important to highlight here that these MBs can be allocated to only one operator at a time. In this way, the NE can have up to four allocated MBs, allowing it to generate sliding windows of 5×9 pixels or 9×9 by associating two consecutive PEs. All data inputs and outputs of the Memory Controller module are connected to the Reconfigurable Interconnection, with the exception of the window output (*oWindow*) connected to the Spatial Processor module (Section 3.1.2) and the configuration input (*pe_cfg*) that is part of the Configuration Tree (Section 3.2). There follows a description of the memory-based operators.

- *Neighborhood Extractor (NE)*: An NE provides a sliding window that scans the whole image. This window contains a pixel neighborhood that is necessary for some computations, e.g., 2D Convolution. The operating principle of a 3×3 neighborhood extractor is presented in Fig. 5. The line buffer size of the NE operator can be configured after synthesis in order to adjust it to the frame size. In addition, the number of line buffers can be configured, from 2 to 4 by allocating more MBs. The number of elements in a pixel array is fixed with nine registers. Thus, the NE operator can support sliding windows of up to 5×9 pixels. The NE operator also supports recursive operations by introducing a second pixel stream directly into the line buffer located just after the central pixel array. This feature allows the PE to process a part of the window twice, what can reduce the number of operators

in certain computations. Also, in order to avoid random pixels in subsequent stages, a border handler is placed at the output of the NE block (Fig. 4). The border handler assigns determined values to pixels of the window that crosses the image's border. This situation occurs when the center of the window is located near to the image border and part of the window stays outside the image boundaries. In our system, we replicate the inner window pixel values to those pixels outside the image boundaries.

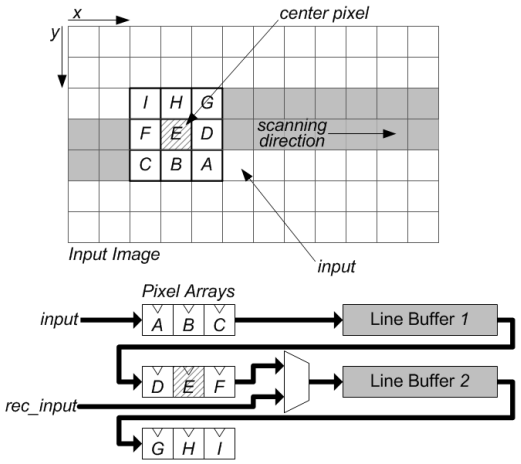


Figure 5: Operating principle of a 3×3 neighborhood extractor.

- *Mirror (Mir*: The function of the Mirror operator is to invert the direction in which the image is scanned. This is necessary when an algorithm must propagate an event in the image and the regular scanning direction does not allow it. The Mirror structure is based on two line buffers working in a Last-In-First-Out (LIFO) fashion.

- *Delay (Z^{-n})*: The Delay operator is composed of two line buffers with configurable size and is used to synchronize two pixel streams that are in different stages of the P^2IP. The maximum delay supported is 2 frame lines which is equivalent to the delay imposed by a 5×5 neighborhood operation.

3.1.2. Image operators

The Spatial Processor is a PE module containing three fundamental neighborhood-based operators for low-level image processing, more specif-

ically, a Two-Dimensional Convolver (2DC), a Non-Maximum Suppressor (NMS), and a connector. Fig. 6 shows the functional diagram of the Spatial Processor module.

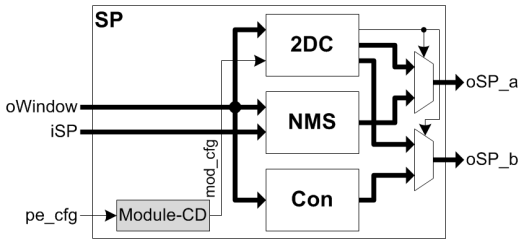


Figure 6: Functional diagram of the Spatial Processor (SP).

The output window from the Memory Controller module (Section 3.1.1) is the main input of the Spatial Processor module. The other inputs and outputs are connected directly to the Reconfigurable Interconnection, with the exception of the configuration input that is part of the Configuration Tree. The 2DC configuration selects which operator will be activated in the module. Next, we present a description of the Spatial Processor operators.

- *Two-Dimensional Convolver (2DC)*: The 2DC is a fundamental operator in many image processing algorithms. It can convolve an input image $f(x, y)$ with a predefined kernel or mask $h(i, j)$ such as Sobel, Gaussian, and Laplacian, as defined in (1).

$$g(x, y) = \sum_{i,j} f(x + i, y + j) \cdot h(i, j) \tag{1}$$

  where $g(x, y)$ is the resulting image, $f(x, y)$ is the input image, and $h(i, j)$ is the kernel. The equation above can be more compactly noted as in (2).

$$g(x, y) = f(x, y) \otimes h(i, j) \tag{2}$$

  A set of these kernel coefficients is stored in a local ROM block. They can be downloaded during the 2DC configuration step. An important feature of the 2DC is that it can compute up to two 3×3 kernels in parallel per module or a single 5×5 kernel. This approach can reduce idle resources in certain operations.

- *Non-Maximum Suppressor (NMS)*: The NMS operator analyses the input window and outputs the center pixel only if it is the local maximum. Otherwise, NMS outputs the value zero. It has a specific input that provides the gradient direction of the input window. This direction is estimated by the direction operator in the Pixel Processor module.

- *Connector (Con)*: The connector operator is used during edge detection to reduce the number of gaps along lines representing edges. It analyses the neighborhood of a pixel to verify if it can be connected to another pixel in its neighborhood.

The Pixel Processor is a PE module containing a set of optimized pixel-to-pixel operators. Fig. 7 depicts the Pixel Processor module showing all internal operators and their respective interconnection. Following, we give an overview of these operators.

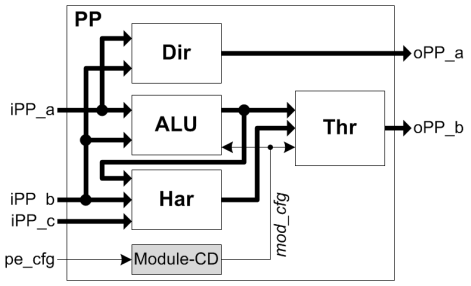


Figure 7: Functional diagram of the Pixel Processor (PP).

- *Direction (Dir)*: This operator compares if the input *iPP_a* value is greater than *iPP_b* and its output (*oPP_a*) is Boolean. It is used as an approximation of the arctangent function which provides the direction of a gradient.

- *Harris (Har)*: The Harris operator computes the Harris response based on [28]. The Harris response is an alternative to calculate the eigenvalues for feature detection applications.

- *Arithmetic and Logic Unit (ALU)*: The ALU operator is a more generic and configurable operator that can compute the following functions: multiplication, square power, binary shifting, addition, subtraction, logic AND (&), greater than, and less than.

- *Threshold (Thr)*: The Threshold operator performs a simple segmentation technique, classifying pixels into two categories according to rule (3)

$$g_{th}(x,y) = \begin{cases} 0, f(x,y) < T \\ 1, f(x,y) \geqslant T \end{cases} \tag{3}$$

where $g_{th}(x,y)$ is the resulting image of the threshold operator, $f(x,y)$ is the image function, and T is the threshold limit value. It can also be configured to operate in a hysteresis mode where two threshold limits (T_{low} and T_{high}) can be used according to rule (4).

$$g_{th}(x,y) = \begin{cases} 0, f(x,y) < T_{low} \\ f(x,y), T_{low} \leqslant f(x,y) < T_{high} \\ 1, f(x,y) \geqslant T_{high} \end{cases} \tag{4}$$

The two clusters of image processing operators in each PE of the P^{2}IP can perform a series of fundamental computations and combinations of them. To illustrate some of the operations supported by the P^{2}IP, Table 1 shows the relations between the P^{2}IP operators and low-level image processing operations.

Table 1: Relation between the P^{2}IP operators and low-level image processing operations.

Low-level Operations	P^{2}IP Operators						
	2DC	NMS	Con	Dir	ALU	Thr	Har
Thresholding						x	
Arithmetic Operations					x		
Smoothing	x						
Sharpening	x				x		
Noise Reduction	x						
Edge Detection	x	x	x	x	x	x	
Corner Detection	x	x			x	x	x
Gradient Direction	x			x			
First-order Derivative	x				x		
Laplacian	x						
Basic Segmentation						x	

3.1.3. Reconfigurable Interconnection

The Reconfigurable Interconnection is the PE module in charge of routing different datapaths in the P^{2}IP architecture. It is composed of two independent configurable crossbars with 8-bit wide paths. Each external input or output of the Reconfigurable Interconnection passes by a register. This approach guarantees that the maximum pixel clock supported by the P^{2}IP does not change with long paths between PEs. Fig. 8 shows the Reconfigurable Interconnection functional diagram.

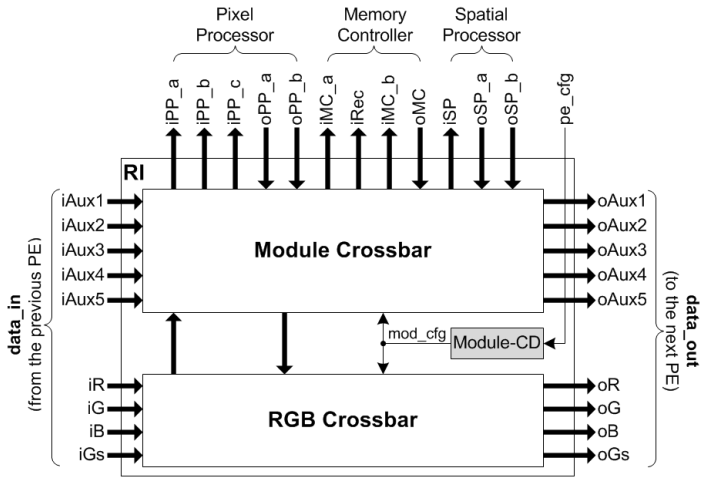


Figure 8: Functional diagram of the Reconfigurable Interconnection (RI).

The module crossbar interconnects all modules in a PE. Its auxiliary I/Os (*iAux* and *oAux*) can share partial results directly with other PEs. The RGB crossbar is the main I/O of a PE. As the P^{2}IP can only process one color component per PE, the RGB crossbar has a single I/O channel connected to the module crossbar. It outputs a selected color component to be processed by the PE and receives the result which is transferred to the next PE. Fig. 9 shows three basic configurations that can be associated to obtain different datapaths.

The pipeline configuration (Fig. 9a) is the natural datapath supported by the P^{2}IP architecture. In this model, the input data passes through a cascaded series of Operators (Op). In the second configuration (Fig. 9b), the input data is processed simultaneously in two different PEs and their outputs are processed in a third PE. The third configuration, presented in Fig. 9c, has three concurrent inputs that are processed simultaneously in three different

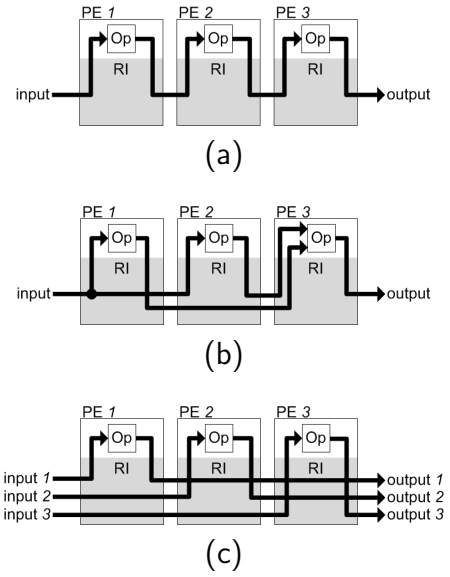


Figure 9: Basic datapath configurations mapped through the Reconfigurable Interconnection (RI). (a) Regular pipeline of Operators (Op). (b) Fork then fusion. (c) Independent parallel pipelines.

PEs. This last configuration is mostly used for color processing.

3.2. Configuration mechanism

The Configuration Tree is a scalable configuration structure formed by a series of Configuration Decoders (CDs) hierarchically organized. The objective of the Configuration Tree is to provide access to all configuration registers located in the P^{2}IP operators. Fig. 10 depicts the Configuration Tree architecture showing all decoder levels, from the configuration input (*config_in*) to the configuration register.

The configuration input of the Configuration Tree is an 8-bit wide bus that can be connected to any kind of external interface, e.g., UART, Ethernet, or flash memory. Differently from the datapath that works at the same clock frequency as the input pixel stream, the Configuration Tree works at a fixed clock frequency of 100 *MHz*. An input configuration word takes four clock cycles to arrive at a specific operator. Each configuration process can send a maximum of eight configuration bytes to a determined operator. Considering the extra two bytes corresponding to the configuration header (operator location and number of extra bytes), an operator takes a maximum of 0.34 μs to be configured.

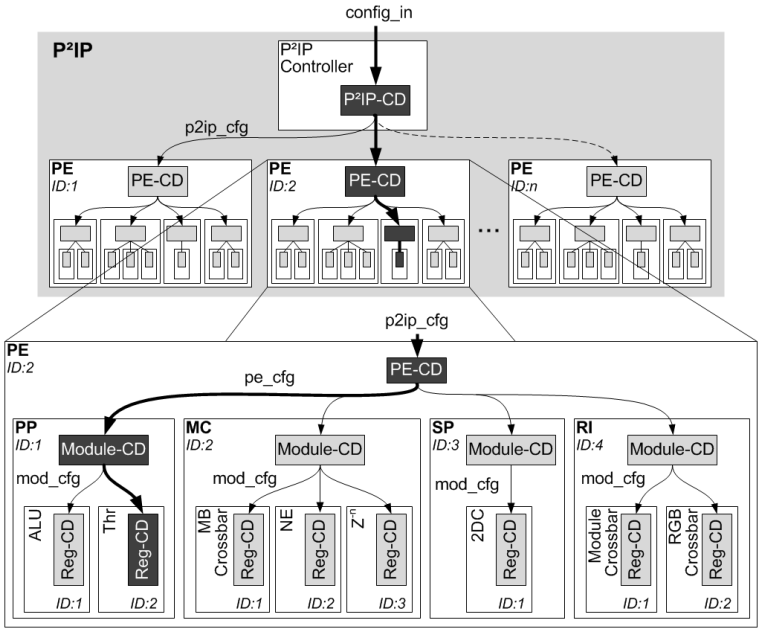


Figure 10: Configuration Tree architecture with a highlighted path showing the active modules during an operator configuration.

The P²IP-CD is the input node of the Configuration Tree. It extracts global parameters from the configuration input, such as the frame size, and transfers the rest to the next level of decoders formed by PE-CDs. Each PE and its internal components have an Identification (ID) number. These IDs are used to identify operators in the P²IP architecture. To access an operator, firstly it is necessary to send a header through the configuration bus containing all IDs in the path towards the operator. During the configuration process, the PE-CDs distribute the configuration words to the next level of decoders inside the processing modules, called Module Configuration Decoders (Module-CD). The Module-CDs send the configuration words to the final level of decoders located inside the processing operators, the Register Configuration Decoders (Reg-CDs). The configuration registers are located in the Reg-CDs and they vary in size depending on how many parameters must be set to configure the operator. Fig. 11 shows the register fields of the configuration header and an example of a configuration register to configure a Threshold operator.

In Fig. 11, the Threshold configuration register is composed by three

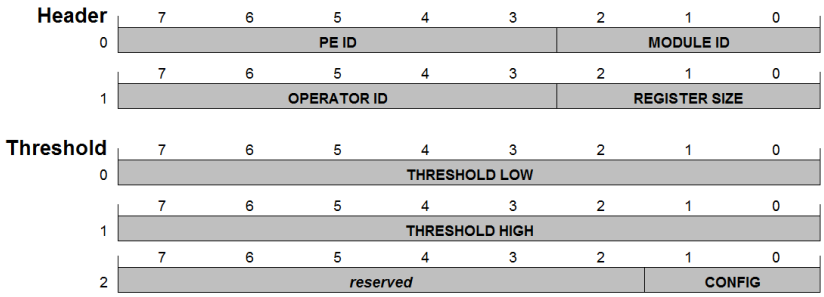


Figure 11: Register fields of the configuration header and the Threshold configuration register.

bytes where the first byte configures the lower threshold limit, the second byte configures the higher threshold limit, and bits 0 and 1 of the third byte indicate the operating mode of the Threshold operator. The Threshold operator can be configured to work in three different modes:

1. Bypass;
2. Normal mode as in eq. (3) where only the lower threshold limit is taken in consideration;
3. Hysteresis mode as in eq. (4) where T_{low} is the lower threshold limit and T_{high} higher threshold limit.

The configuration transfer at the configuration input port (*config_in*) is accepted when the corresponding *VALID* signal is high. Fig. 12 presents an example of configuration transfer.

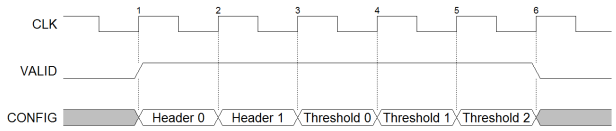


Figure 12: Example of configuration transfer targeting a Threshold operator.

The methodology to configure the P^{2}IP by manually creating configuration transfer sequences is timing consuming and error prone. In order to facilitate the P^{2}IP configuration process, a function library embedded on MATLAB-based language has been created. This library accepts mnemonics to cover all possible configurations of the P^{2}IP operators, giving to the user full control over the architecture. In addition to creating configuration transfer sequences, MATLAB allows the user to send them directly to the

P^{2}IP by creating *Interface Objects* (e.g. UART, UDP, I^{2}C, and Bluetooth). Following, we give an example of configuration using the function library:

```
>> P2IP(ser, 'PE',2, 'threshold', 'Tcfg',3, ...
   'Ta',50, 'Tb',100)
```

where *P2IP(...)* is the main MATLAB function that calls the internal functions; *ser* is a MATLAB opened communication port that will be used to transfer the configuration words generated by the assembler; *'PE'* is the mnemonic for PE ID that receives the value 2; and *'threshold'* is the mnemonic for the threshold operator that receives three parameters: $Tcfg = 3$ corresponding to the hysteresis operating mode; $Ta = 50$ corresponding to T_{low} value; and $Tb = 100$ corresponding to T_{high} value.

4. Algorithm partitioning and allocation

The objective of algorithm partitioning is to divide the algorithm into groups of operations that can fit in a single PE and analyze the appropriate configuration of the interconnection. The basic rule for an efficient algorithm mapping on the P^{2}IP is to allocate at least one neighborhood operator per PE and distribute the remaining operators according to the available resources and their data dependencies.

In order to validate the proposed architecture, three low-level image processing algorithms have been mapped on the P^{2}IP, more specifically, the Edge Sharpening, Canny Edge Detection, and Harris Corner Detection algorithms. These algorithms are very popular in the field and used in a wide range of applications, e.g. image enhancement, image segmentation, object recognition, and object tracking. In addition, they require a variety of operators that offer an interesting use case for evaluating our approach. Following, we give an overview of these application algorithms and then a description of the partitioning of each algorithm.

4.1. Edge Sharpening

Edge Sharpening is a technique for enhancing edges in images with a low level of edge definition, e.g. X-ray images. It commonly uses the unsharp sharpening algorithm where an unsharp filter extracts the image gradient edges and then adds them back onto the original image [29]. Mathematically,

the unsharp filter produces an edge image $e(x, y)$ from an input image $f(x, y)$, as defined in (5).

$$e(x, y) = f(x, y) - f_{smooth}(x, y) \tag{5}$$

where $f_{smooth}(x, y)$ is a smoothed version of $f(x, y)$ that must be previously computed.

To obtain the sharpened edges, the result of (5) is added back to the original image, as in (6).

$$s(x, y) = f(x, y) + k \cdot e(x, y) \tag{6}$$

where $s(x, y)$ is the resulting image with sharpened edges, and k is a scaling constant.

A different commonly used technique is to apply a negative Laplacian kernel (7) as the unsharp filter, simplifying the algorithm in one operation. Using this technique, the unsharp filter $e(x, y)$ can be defined as in (8).

$$h_l(i, j) = \frac{1}{16} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{7}$$

$$e(x, y) = f(x, y) \otimes h_l(i, j) \tag{8}$$

Fig. 13 presents the Edge Sharpening algorithm graph for a single color channel applying a Laplacian kernel (h_l) as the unsharp filter and its implementation on the P^{2}IP architecture in a true-color (24-bit RGB) format.

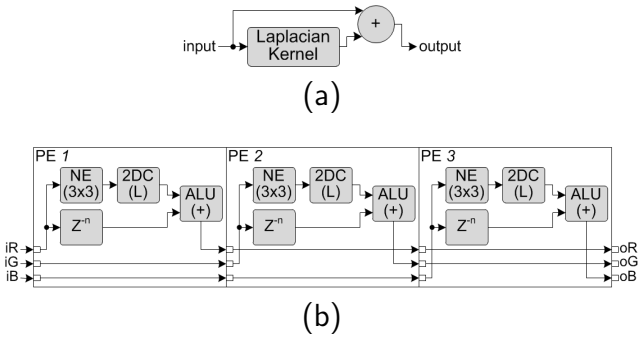


Figure 13: Edge Sharpening algorithm. (a) Algorithm graph. (b) P^{2}IP implementation.

The Edge Sharpening implementation (Fig. 13b) requires three PEs for a true-color format and performs 60 operations per pixel along the datapath.

The datapath configuration is similar to the one presented in Fig. 9c where each PE has an identical set of operators working in parallel. For each color channel, the input is computed by a 3×3 Laplacian (L) kernel and the ALU operator adds its result to a delayed version of the input provided by the delay (Z^{-n}) operator.

4.2. Canny Edge Detection

Edge detection is a fundamental process in computer vision for image segmentation and object recognition. Edges are prominent events due to local changes in intensity or color in images [30]. Basically, if the brightness of a pixel is significantly different from the pixels in its neighborhood, it may contain an edge. It is possible to detect such changes in an image $f(x, y)$ by applying special kernels, such as the Sobel kernel (9). These kernels are presented in pairs where one is used to detect horizontal variations (h_h) on the image and the other vertical variations (h_v). As a result of these operations, we obtain a horizontal gradient $g_h(x, y)$ and a vertical gradient $g_v(x, y)$ as defined in (10) and (11), respectively.

$$h_h(i,j) = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } h_v(i,j) = \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{9}$$

$$g_h(x,y) = f(x,y) \otimes h_h(i,j) \tag{10}$$

$$g_v(x,y) = f(x,y) \otimes h_v(i,j) \tag{11}$$

With these gradients, it is possible to obtain a gradient magnitude or modulus that represents the local edge strength $|G(x, y)|$ (12) and the local edge orientation angle Φ (13) using an arctangent operation.

$$|G(x,y)| = \sqrt{g_h(x,y)^2 + g_v(x,y)^2} \tag{12}$$

$$\Phi(x,y) = tan^{-1}\left(\frac{g_v(x,y)}{g_h(x,y)}\right) \tag{13}$$

Canny Edge Detection is one of the most popular algorithms for edge detection due to its minimum number of false edge points, good localization of edges, and single mark on each edge [30]. The Canny algorithm is composed of three steps: smoothing, edge enhancement, and localization. For the

smoothing step, the Canny algorithm uses a Gaussian low pass filter, based on a Gaussian kernel (14), to suppress the noise of the input image.

$$h_g = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{14}$$

Next, during the edge enhancement step, a gradient vector at each pixel of the smoothed image is calculated based on the Sobel kernel, as defined in (9). The localization step is divided into two stages: NMS and Hysteresis Thresholding (HThr). The objective of the NMS is to eliminate non-ridge pixels giving a one pixel wide aspect at the edges. A ridge pixel is defined as a pixel with a gradient magnitude greater than that of the adjacent pixels in the direction of the gradient. In the HThr stage, two thresholds are used: T_{low} and T_{high}. All pixels with a magnitude higher than T_{high} are considered as true edges. Pixels with magnitudes between T_{high} and T_{low} are considered as edge candidates. Pixels that do not satisfy these two criteria are suppressed. Edge candidates become true edges if they are connected to true edges directly or through other candidates. Fig. 14a presents the algorithm graph of the Canny Edge Detection, while Fig. 14b shows how the algorithm is partitioned and implemented on the P^{2}IP architecture.

The Canny Edge Detection implementation (Fig. 14b) works with a flow of 8-bit grayscale pixels. It starts by computing a 5×5 Gaussian (G) kernel. In *PE2* two 3×3 kernels are computed in parallel, the horizontal Sobel (S_H) and the vertical Sobel (S_V). Also in this PE, the gradient direction is estimated by the direction (Dir) operator and the ALU computes a gradient magnitude approximation based on [29]. The direction is transferred by an auxiliary channel (*oAux1*) to the next PE where it is used to process the NMS. Finally, the threshold (Thr) operator of the *PE3* along with *PE4* and *PE5* perform an approximation of the HThr operation where a sequence of connectors (Con) and mirrors (Mir) can fill most of the gaps in edge lines. The datapath structure in this implementation is mostly a regular pipeline as presented in Fig. 9a. The mapped version occupies five PEs and performs 123 operations per pixel along the datapath.

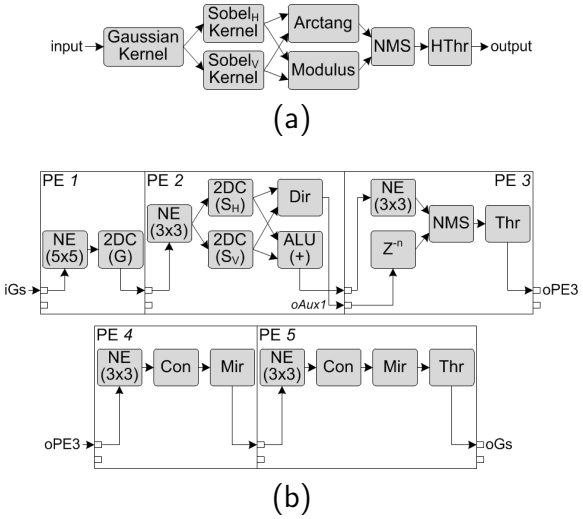


Figure 14: Canny Edge Detection algorithm. (a) Algorithm graph. (b) P^{2}IP implementation.

4.3. Harris Corner Detection

Corner detection is another fundamental process in computer vision. It is mainly used for motion detection, object tracking, panorama stitching, and 3D modeling. A corner is defined as an area that exhibits a strong gradient value in multiple directions at the same time [30]. The Harris operator uses this premise to find corners on an image. The first step is to obtain the first partial derivative of the image function $f(x, y)$ in both directions, horizontal and vertical, based on approximations (15) and (16).

$$g_H(x, y) = \frac{\delta f}{\delta x}(x, y) \approx f(x, y) \otimes \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{15}$$

$$g_V(x, y) = \frac{\delta f}{\delta y}(x, y) \approx f(x, y) \otimes \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{16}$$

With the values of $g_H(x, y)$ and $g_V(x, y)$, it is possible to calculate the elements of the matrix M, described in (17), using (18), (19), and (20).

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{17}$$

$$A = g_H(x, y)^2 \otimes w \tag{18}$$

$$B = g_V(x,y)^2 \otimes w \tag{19}$$

$$C = (g_H(x,y) \cdot g_V(x,y)) \otimes w \tag{20}$$

where w is a smoothing circular operator, e.g., a Gaussian kernel as defined in (14). The final step is to obtain the Harris operator response as in (21).

$$R = \mathrm{Det}[M] - k \cdot \mathrm{Tr}^2[M] \tag{21}$$

where R is positive in corner regions, negative in edge regions, and small in flat regions, k is a constant coefficient that, in practice, is a fixed value in the range of 0.04 to 0.06, and Det and Tr are the *determinant* and *trace* matrix operations, respectively. An optional final step is to select the best results in a determined region in order to reduce false corners. It can be done by a 10×10 NMS operator. Fig. 15a presents the algorithm graph of the Harris Corner Detection, while Fig. 15b shows how the algorithm is partitioned and implemented on the P^{2}IP architecture.

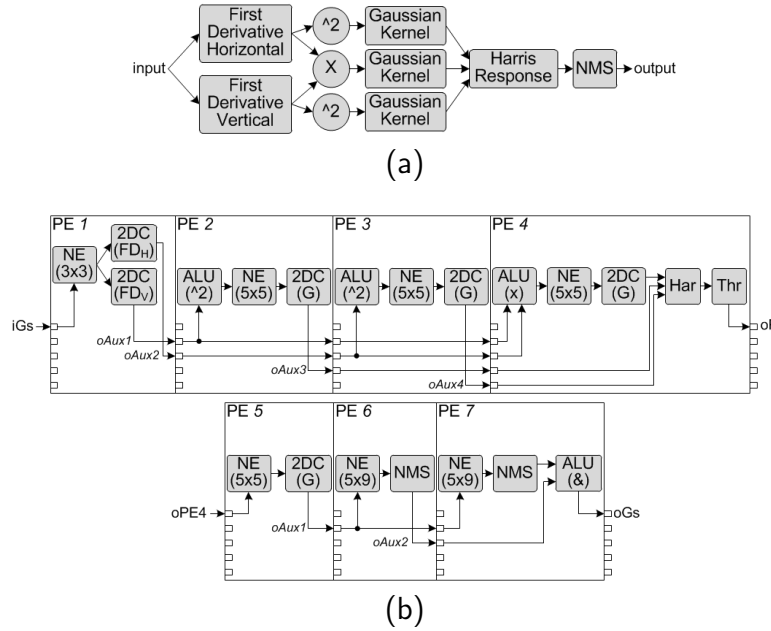


Figure 15: Harris Corner Detection algorithm. (a) Algorithm graph. (b) P^{2}IP implementation.

The Harris Corner Detection implementation on the P^{2}IP architecture (Fig. 15b) works with a flow of 8-bit grayscale pixels. It starts by computing

two 3×3 kernels in parallel, the horizontal first derivative (FD_H) and the vertical first derivative (FD_V). The output of the *PE1* is transferred to 3 different PEs (2, 3, and 4). In *PE4*, the Harris (H) operator uses the result of the previous 5×5 Gaussian kernel along with the results from *PE2* and *PE3* to compute the Harris response. The datapath structure along *PE2*, *PE3*, and *PE4* is an example of the model presented in Fig. 9b. In *PE5*, an extra 5×5 Gaussian kernel is computed in order to help the NMS operator during the selection of the best results. Finally, *PE6* and *PE7* create two 5×9 windows where one represents the top and the other represents the bottom part of a 9×9 window. The P^{2}IP version of the algorithm requires 7 PEs and performs 340 operations per pixel along the datapath.

5. Results

As mentioned earlier, latency is a critical characteristic of image processing systems in applications that must react as fast as possible to events captured by the image sensor. In the P^{2}IP, each PE can have a different latency according to its configuration. In order to simplify the latency analysis, we will assume a worst-case scenario, when a pixel stream takes the longest path along a PE. The longest path in a PE is shown in Fig. 16 where the corresponding latency is expressed in pixels since the P^{2}IP works at the input stream clock frequency, processing one pixel per clock cycle.

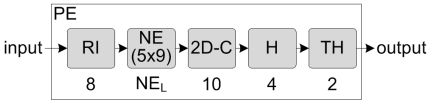


Figure 16: Longest path in a PE and the related latency. The latency is expressed in pixels.

The NE latency (NE_L) for a neighborhood window with $m \times n$ pixels in an image with dimensions $M \times N$ pixels, can be expressed as defined in (22).

$$NE_L = N\left(\frac{m-1}{2}\right) + \frac{n-1}{2} + b \tag{22}$$

where N is the number of pixels in one line of the input image and, for the neighborhood window, m is the number of lines, n is the number of pixels per line, and b is the Border Handler latency. As the largest window provided by

a single NE is 5×9 and the border handler latency is 3 clock cycles, we can obtain the PE total latency for the worst-case scenario (PE_L) as in (23).

$$PE_L = 2N + 32 \tag{23}$$

Finally, the total latency of the system can be expressed as in (24), taking into consideration that PEs operating in parallel are not included as active PEs.

$$P^2IP_L = p_a \cdot PE_L \tag{24}$$

where p_a is the number of active PEs in the P^{2}IP architecture.

In order to evaluate the P^{2}IP performance, we have implemented the architecture containing 10 PEs and supporting line buffers of up to 4095 pixels in an FPGA-based platform Altera Stratix IV EP4SGX230. The maximum clock frequency (F_{max}) in the datapath reported after synthesis was 268 *MHz*. As the P^{2}IP delivers one processed pixel per clock cycle, the datapath F_{max} corresponds to the architecture throughput that in this case is 268 Mpixel/s. Table 2 presents the P^{2}IP benchmarking obtained from HDL simulation targeting the low-level image processing algorithms described in the last section. Due to the constant output data rate, the throughput results do not change from one application to another. The performance varies according to the number of operations per pixel along the datapath, but stays constant for any resolution considering maximum throughput. Fig. 17 shows examples of images processed by the P^{2}IP using the algorithms mentioned.

Table 2: Throughput (T), Performance (P), and Latency (L) benchmarking results for Edge Sharpening (ES), Canny Edge Detection (CED), and Harris Corner Detection (HCD) algorithms w.r.t. P^{2}IP (10 PEs @ 268 *MHz*).

Application	Full HD[a]		4K[b]		Any
	T (fps)	L (μs)	T (fps)	L (μs)	P ($GOPS$)
ES	129	21.9	32	43.3	16.1
CED		43.5		86.6	33.0
HCD		65.1		129.5	91.1

[a] 1080×1920 pixels.
[b] 2160×3840 pixels.

Table 3 shows the FPGA resources required by the P^{2}IP architecture with 10 PEs. Considering that the P^{2}IP reconfigurability is mainly supported

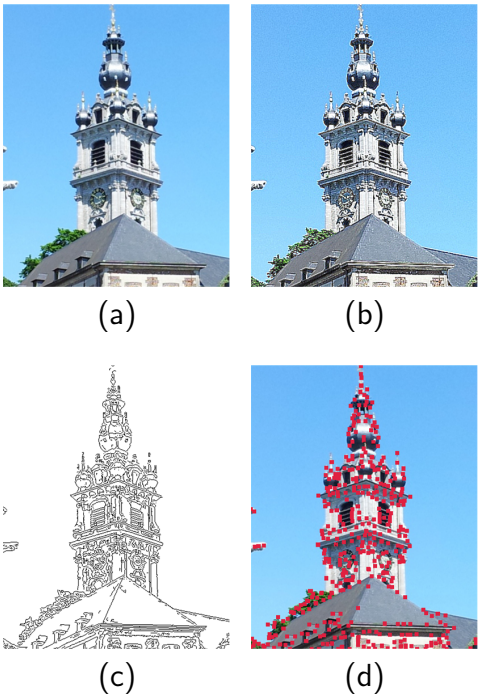


Figure 17: Example of images processed on the P^{2}IP architecture. (a) Original image - The belfry of Mons. (b) Edge Sharpening. (c) Canny Edge Detection - inverted output for better visualization. (d) Harris Corner Detection.

by the Configuration Tree and the Reconfigurable Interconnection, we can establish the amount of resource overhead generated only by this feature, corresponding to 15 % of the total resources. This overhead is more important when we take into consideration the resources left idle in comparison with a dedicate not-flexible implementation of the same algorithm. An example of this is when we compare a bare-metal implementation of the Harris Corner Detection with its analogous P^{2}IP implementation. The overhead in this case is an average of 74 % in each PE. If resource utilization is a design concern, the overhead can be drastically reduced by mapping as fixed PEs' parameters, all operators that are not required to be modified after-synthesis. During synthesis, the synthesizer in capable of optimizing these PEs by removing all resources that are not associated to the mapped operators, i.e., idle resources.

A comparison between the P^{2}IP performance and the state-of-the-art architectures is presented in Table 4. We can see that P^{2}IP has a better

Table 3: FPGA resources w.r.t. P^{2}IP (10 PEs).

Component	ALM[a]	Dedicated registers	Memory ($kbits$)	DSPs[b]
Controller	218	133	0	0
Input Reg.	31	50	0	0
Output Reg.	139	76	0	0
PE	3028	3766	131	29
Total	3416	4025	131	29
Total (10 PEs)[c]	30668	37919	1310	290

[a] Adaptive Logic Module (ALM).
[b] DSP block elements distributed on the FPGA fabric.
[c] Total resources with 10 PEs.

performance than all architectures presented in Table 4. P^{2}IP has shown a performance enhancement from 1.7 to 3.18 times faster than the state-of-the-art related works. In terms of latency, it is clear that P^{2}IP has a big advantage over the architectures that have addressed this specification.

To obtain this comparison, we have taken applications and parameters (image size) presented in [21, 14, 22, 13, 31] and similarly mapped them onto the P^{2}IP. This gave us a fair basis to compare their results against the results obtained with the P^{2}IP. As presented earlier, our proposed architecture supports any image size up to 4095×4095 pixels, only limited by the size of the line buffers present in the PEs.

The throughput of the P^{2}IP is highly dependent on the F_{max} obtained during synthesis. The obtained F_{max} will certainly be different depending on the technology where the architecture is implemented (e.g., different FPGAs vendors or families and different ASIC technology). The results presented in this section attempt to illustrate the P^{2}IP performance on a relatively old FPGA technology (40 nm process technology) which we have had available for prototyping.

6. Conclusions

In this paper, we have presented P^{2}IP, a novel coarse-grained reconfigurable systolic array for real-time image and video processing. With a run-time reconfigurable datapath associated to a dedicated programming mechanism, the P^{2}IP architecture can perform many low-level image processing

Table 4: Throughput (T) and operating frequency (F) comparison with the state-of-the-art architectures w.r.t. P^{2}IP (10 PEs) @ 268 *MHz*.

					P^{2}IP
		Image Size	F	T	T
Architecture	Application	($M \times N$ *pixels*)	(MHz)	(fps)	(fps)
CSX700 [21]	HCD[a]	720×1280	250	91	290
Diet SODA [14]	2D Filter	2720×4072	400/50[b]	13	24
CRISP [22]	2D Filter	2720×4072	115	9	24
GPU [13]	CED[c]	3936×3936	575	10	17
Hybrid [31]	CED	1472×1760	2400/1296[d]	47	103

[a] Harris Corner Detection.
[b] 400 *MHz* is the frequency of the memory and controller systems and 50 *MHz* is the SIMD frequency.
[c] Canny Edge Detection.
[d] 2400 *MHz* is the CPU frequency and 1296 *MHz* is the GPU frequency.

applications.

In our study, three image processing algorithms were mapped on the proposed architecture in order to validate it. Reliable output results were achieved and discussed for that purpose. We also show how each algorithm is mapped on the architecture via the P^{2}IP configuration mechanism on different PEs all configured to become one application. For different image sizes, we show the latency of our proposed architecture and compared to that of the state-of-the-art architectures, presenting very competitive performances with a throughput of one processed pixel per clock cycle independently of the operating frequency. These obtained results proved that the proposed architecture can be a suitable low-level image processor candidate for commercial embedded systems targeting modern video standards. Finally, from a System-on-Chip point of view, the P^{2}IP can be seen as a new flexible building block with an industrial standard interface base on the AXI4-Stream interconnection protocol.

Regarding future extensions of the present work, an important addition could be an evolution of the MATLAB-based function library by using higher-level entry methods such as MATLAB/Simulink or LabVIEW. Furthermore, the addition of an expert system capable of inferring routing solutions and adjusting internal delays based on a desired P^{2}IP configuration could be very beneficial to the architecture acceptance. This expert system could

make the internal routing and delay balancing transparent for the user and complemented by a higher level entry method.

References

[1] T. R. Savarimuthu, A. Kjær-Nielsen, A. S. Sørensen, Real-time medical video processing, enabled by hardware accelerated correlations, Journal of Real-Time Image Processing 6 (2011) 187–197. doi:10.1007/s11554-010-0185-2.

[2] M. Gorgoń, Parallel performance of the fine-grain pipeline FPGA image processing system, Opto-Electronics Review 20 (2012) 153–158. doi:10.2478/s11772-012-0021-2.

[3] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, A. Das, Evaluating the Imagine stream architecture, in: Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA04), 2004.

[4] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, J. D. Owens, Programmable stream processors, Computer (2003) 54–61.

[5] IEEE, IEEE Standard Glossary of Computer Hardware Terminology, IEEE Computer Society, 1994.

[6] A. P. Reeves, Parallel computer architectures for image processing, Computer Vision, Graphics, and Image Processing 25 (1) (1984) 68–88. doi:10.1016/0734-189X(84)90049-5.

[7] K. Diefendorff, P. K. Dubey, How multimedia workloads will change processor design, Computer (1997) 43–45.

[8] B. H. McCormick, The Illinois pattern recognition computer-ILLIAC III, IEEE Transactions on Electronic Computers EC-12 (6) (1963) 791–813. doi:10.1109/PGEC.1963.263562.

[9] B. K. Khailany, T. Williams, J. Lin, E. P. Long, M. Rygh, D. W. Tovey, W. J. Dally, A programmable 512 GOPS stream processor for signal, image, and video processing, IEEE Journal of Solid-state Circuits 43 (1) (2008) 202–213.

[10] C. Bobda, R. Hartenstein, Introduction to reconfigurable computing : architectures, algorithms, and applications, Springer-Verlag, Dordrecht, 2007.

[11] D. Baumgartner, P. Roessler, W. Kubinger, C. Zinner, K. Ambrosch, Benchmarks of low-level vision algorithms for DSP, FPGA, and mobile PC processors, in: B. Kisacanin, S. Bhattacharyya, S. Chai (Eds.), Embedded Computer Vision, Advances in Pattern Recognition, Springer London, 2009, pp. 101–120. doi:10.1007/978-1-84800-304-05.

[12] K. Illgner, DSPs for image and video processing, Signal Processing 80 (11) (2000) 2323 – 2336, Special section on DSP in Audio-visual communications. doi:10.1016/S0165-1684(00)00120-1.

[13] Y. Luo, R. Duraiswami, Canny edge detection on NVIDIA CUDA, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2008, pp. 1–8. doi:10.1109/CVPRW.2008.4563088.

[14] S. Seo, R. G. Dreslinski, M. Woh, C. Chakrabarti, S. Mahlke, T. Mudge, Diet SODA: a power-efficient processor for digital cameras, in: Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design, ISLPED '10, ACM, New York, NY, USA, 2010, pp. 79–84. doi:10.1145/1840845.1840862.

[15] S. Vassiliadis, D. Soudris (Eds.), Fine- and Coarse-Grain Reconfigurable Computing, Springer, 2007.

[16] S. Hauck, A. DeHon (Eds.), Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[17] N. S. Voros, A. Rosti, M. Hbner, Dynamic System Reconfiguration in Heterogeneous Platforms: The MORPHEUS Approach, Vol. 40 of Lecture Notes in Electrical Engineering, Springer Netherlands, 2009. doi:10.1007/978-90-481-2427-5.

[18] J. M. P. Cardoso, M. Hübner (Eds.), Reconfigurable Computing: From FPGAs to Hardware/Software Codesign, Springer, 2011. doi:10.1007/978-1-4614-0061-5.

[19] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, R. R. Taylor, PipeRench: A virtualized programmable datapath in 0.18 micron technology, in: IEEE Custom Integrated Circuits Conference (CICC), 2002.

[20] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, The Raw microprocessor: A computational fabric for software circuits and general purpose programs, IEEE Micro (2002) 25–35.

[21] A. Fijany, F. Hosseini, Image processing applications on a low power highly parallel SIMD architecture, in: IEEE Aerospace Conference, 2011, pp. 1–12. doi:10.1109/AERO.2011.5747456.

[22] J. Chen, S.-Y. Chien, CRISP: Coarse-grained reconfigurable image stream processor for digital still cameras and camcorders, IEEE Transactions on Circuits and Systems for Video Technology 18 (9) (2008) 1223–1236. doi:10.1109/TCSVT.2008.928529.

[23] S. Mahmoudi, P. Manneback, C. Augonnet, S. Thibault, Détection optimale des coins et contours dans des bases d'images volumineuses sur architectures multicœurs hétérogènes, in: 20ème Rencontres francophones du parallélisme (RenPar'20), 2011.

[24] ARM, AMBA open specifications (Jan. 2013).
URL `www.arm.com`

[25] Xilinx, Hierarchical design methodology guide (Mar. 2011).
URL `www.xilinx.com`

[26] Altera, Quartus II Handbook v12.1 (Nov. 2012).
URL `www.altera.com`

[27] C. Poynton, Digital Video and HDTV Algorithms and Interfaces, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[28] C. Harris, M. Stephens, A combined corner and edge detection, in: Proceedings of The Fourth Alvey Vision Conference, 1988, pp. 147–151.

[29] T. B. Moeslund, Introduction to Video and Image Processing, Springer, 2012.

[30] W. Burger, M. J. Burge, Digital Image Processing: An Algorithmic Introduction using Java, Texts in Computer Science, Springer London, 2008.

[31] F. Lecron, S. A. Mahmoudi, M. Benjelloun, S. Mahmoudi, P. Manneback, Heterogeneous computing for vertebra detection and segmentation in X-ray images, International Journal of Biomedical Imaging Volume 2011 (2011) pp. 1–12. doi:10.1155/2011/640208.

Paulo Possa received his B.Sc. degree in Electronics Engineering from the University of Passo Fundo, Brazil, in 2005. In 2008, he received his M.Sc. degree in Biomedical Engineering from the Federal University of Santa Catarina, Brazil. He received his Ph.D. degree in Engineering Science from the University of Mons, Belgium, in 2013. His main research interests are reconfigurable computing, real-time embedded systems, digital image processing, and HDL and hardware design.

Naim Harb is a senior researcher for the Electronics and Microelectronics Department of the University of Mons, Belgium. He received his BSc degree in Computer and Communication Engineering from the Islamic University of Lebanon, Lebanon, in 2005. In 2008, he received his Engineering Master's degree in Electrical and Computer Engineering from the American University of Beirut, Lebanon. He obtained a Ph.D. degree in Computer Science from the University of Valenciennes, France, in 2011. His research interests are embedded systems, FPGAs, partial and dynamic reconfiguration, image and signal processing, bioinformatics, hardware system design and optimization.

Eva Dokládalová received her Ph.D. degree in Mathematical Morphology at the Ecole des Mines de Paris (currently Mines-ParisTech) in 2004. After the thesis, she joined the Laboratory of Embedded Computers and Image at the Atomic Energy Commission in Saclay, where she contributed to research projects in the field of embedded architectures for image and video compression. Currently, she is a Professor-Researcher at the Computer Science Department of the ESIEE-Paris. Her research interests are focused on hardware architectures for advanced real-time embedded vision systems and image processing algorithms for embedded vision.

Carlos Valderrama is the director of the Microelectronics Department at the University of Mons, Belgium, which is dedicated to the design of integrated electronic circuits and digital systems. Previously, he was leading projects in CoWare in Belgium, and a Visiting Professor at Brazilian Universities. He received the EE engineering diploma from the National University of Cordoba, Argentina, in 1989. In 1993, he received the MSc in Microelectronics from the Federal University of Rio de Janeiro, Brazil. He obtained his Ph.D. degree from the Grenoble Institute of Technology, France, in 1998.

Unsupervised Perception Model for UAVs Landing Target Detection and Recognition

Eric Bazán[1], Petr Dokládal[1], and Eva Dokládalová[2]

[1] PSL Research University - MINES ParisTech, CMM - Center for Mathematical Morphology, Mathematics and Systems, 35, rue St. Honoré, 77305, Fontainebleau Cedex, France
{eric.bazan, petr.dokladal}@mines-paristech.fr
[2] IGM, Unité mixte de recherche CNRS-UMLV-ESIEE, UMR 8049, Université Paris-Est, Cité Descartes B.P.99, 93162, Noisy le Grand Cedex, France
eva.dokladalova@esiee.fr

Abstract. Today, unmanned aerial vehicles (UAV) play an interesting role in the so-called Industry 4.0. One of many problems studied by companies and research groups are the sensing of the environment intelligently. In this context, we tackle the problem of autonomous landing, and more precisely, the robust detection and recognition of a unique landing target in an outdoor environment. The challenge is how to deal with images under non-controlled light conditions impacted by shadows, change of scale, perspective, vibrations, noise, blur, among others. In this paper, we introduce a robust unsupervised model allowing to detect and recognize a target, in a perceptual-inspired manner, using the Gestalt principles of non-accidentalness and grouping. Our model extracts the landing target contours as outliers using the RX anomaly detector and computing proximity and a similarity measure. Finally, we show the use of error correction Hamming code to reduce the recognition errors.

Keywords: UAV, landing target, perception model, object detection, precision landing

1 Introduction

In this paper, we present a novel method for the detection of landing targets for the UAV vision aided landing. We propose to model the landing target by taking into account the principles of the human perception. The methodology presented works in an unsupervised mode, i.e., no need to adjust parameters.

In outdoor environments, many variables affect the vision-based landing target detection. The main problems to face are: the non-controlled light changes that generate shadowing, reflectance and saturation on the surfaces; the perspective and distance of the camera that deforms the objects; the motions and vibrations that blur the images and; the noise generation by a low-quality sensor.

The detection of the landing target can be viewed as an image segmentation problem, where there is a wide range of developed methods. The variational

framework [13], offers an optimal general method for image segmentation; however, its mathematical complexity and the constant selection of fidelity and a regularization parameters makes its use complex. Also, the number of iterations needed to find the optimal solution avoid having results in real-time. Conversely, thresholding methods have been used for the detection of landing targets [8][9] for its ease of use. However, for a good detection, its use is limited to indoor spaces, where the light conditions are controlled [1].

Recently, convolutional neural networks (CNN) techniques offer the possibility of detecting an object from a large set of classes with a high-reliability [3]. Nevertheless, these methods must have been trained with a database containing the object classes in a wide range of situations and, in case of changes in the object or the scene, the database must be rebuilt [19][6]. Besides, in some cases, the computation is carried out off-board the drone, which implies the need for network infrastructure and limitation of autonomy [10].

Humans can carry out the process of perception in a natural way [14]. We identify meaningful features and exciting events in a scene (such as points, lines, edges, textures, colors, movement) and with the help of our memory and the learning capacity we can recognize and classify objects. The primitives identification is a consequence of their non-accidental apparition, i.e., they are not generated randomly [2]. The Gestalt theory [17] states that we can build a whole (gestalt) through the grouping of non-accidental detected primitives. In this work, we explore the above ideas and propose a novel approach to detect a landing target in the same way as humans do, imitating the human perception process.

The work is organized as follows. In section 2 we develop the perception model. Specifically, subsection 2.2 describes how to retrieve image contours as meaningful primitives and subsection 2.3 describes how to group the contours to detect a landing target perceptually. Later, in section 3 we present the landing target design and the technique used for the methodology implementation; the results obtained are discussed in subsection 3.2. Finally, we present some conclusion and perspectives in section 4.

2 The Unsupervised Perception Model

2.1 Outset Presentation

The algorithm uses the contours as image primitives to obtain information about the scene. For the detection and recognition of landing targets, the algorithm is divided into three major stages. The first, localize all the image contours and extract meaningful contours using the non-accidentalness principle. The second stage computes some feature contours and, based on them, group the meaningful contours through the Gestalt laws. The last stage performs the target recognition using a decoding technique described in section 3. Figure 1 shows the three major stages and its subtasks.

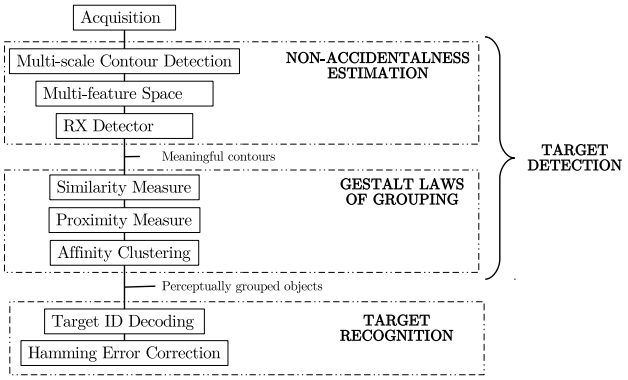


Fig. 1: Diagram of the phases for the landing target detection and recognition

2.2 Non-accidentalness Estimation

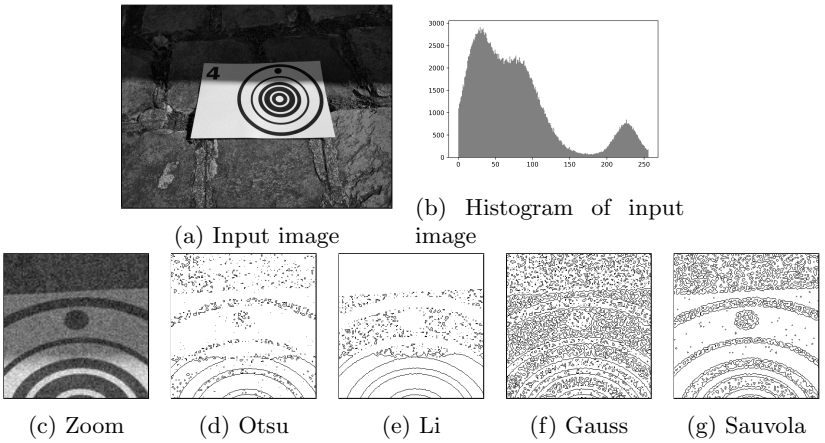
We aim to detect object contours in natural images where none, one or more landing targets can be present. Due to its real-time capacity, it is tempting to use a thresholding method to detect the contours of a binary image. We implemented several thresholding methods analyzed in [16], however, given the conditions where a landing target can be found, no method was found robust enough to variations in non-controlled outdoor environments. Figure 2a shows a landing target in an outdoor environment; we also show his histogram to highlight the levels of saturation in the scene. As a comparison, we take one representative method of each class of the taxonomy proposed in [16] to extract the contours of the image; clustering-based (fig. 2d), entropy-based (fig. 2e), spacial (fig. 2f) and local (fig. 2g) thresholding methods. Namely, there is no guarantee that the contours found by thresholding are present and continuous alongside the object borders.

Contour Detection Instead of using a thresholding method, we obtain the image without fixing any parameter. The use of the Marr-Hildreth [12] operator guarantees to find continuous and closed contours eliminating the possible noise in the image, while the contours of objects remain unchanged in the presence of shadows. This technique convolves the intensity image f with the 2-D Laplacian of Gaussian operator $\nabla^2 G(x, y, \sigma)$ and generates an image,

$$l_\sigma = \nabla^2 G(\sigma) * f \tag{1}$$

in which we localize the zero-crossings.

The parameter σ in eq. (1) permits to control the amount of image smoothing, but also acts as scale parameter, that when varies, it generates different scale-space images. Since no single filter can be optimal simultaneously at all scales [12], we use a multi-scale analysis [18] to detect the zero-crossings in l_σ at different scale-spaces to minimize the risk that some contour of interest is not detected.



(a) Input image (b) Histogram of input image

(c) Zoom (d) Otsu (e) Li (f) Gauss (g) Sauvola

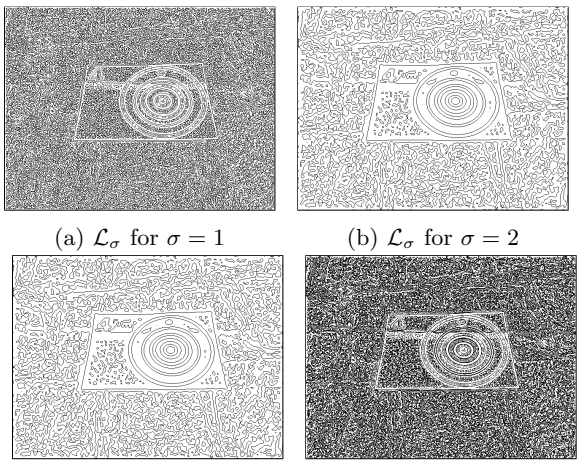
Fig. 2: Landing target under non-controlled illumination conditions and the controus obtained with some thresholding methods

The image l_σ from eq. (1) contains a set of contours $\mathcal{L}_\sigma = \{L_i^\sigma,\ \ i = 0, 1, \ldots, N\}$ for a given scale σ. Then,

$$\mathcal{L} = \bigcup_{\sigma} L_\sigma \tag{2}$$

represents all the contours of an image obtained at different scale-spaces. Figure 3d shows the set of contours $\mathcal{L}$ found for $\sigma = [1, 2, 3]$. Besides, it is also appreciated that at a fine scale (Fig. 3a) we can see more characteristics of the objects, i.e., there are more contours. Conversely, in coarse scales (Fig. 3c), due to the smoothing, there is a spatial distortion, and fewer contours appear. However, those contours that had already appeared at a coarse scale, will not disappear. Then, exist the probability that those contours that spatially coincide on two or more scales belong to a change of intensity generated by the border of an object.

Multi-feature Space The Helmholtz principle states that meaningful characteristics appear as large deviations from randomness and that is how the human perception automatically works to identify an object [2]. The a contrario model proposed in [4], formulates this principle statistically by setting the number of false alarms (NFA) below some acceptable level; however, this method cannot be easily extended to more complex shapes. Instead of setting the NFA, we use the RX detector [15] to detect outliers. Initially called the constant false alarms rate detection algorithm (CFAR) it can detect the presence of a know signal pattern in several signal-plus-noise channels. For that, it uses a $N \times Q$ multi-variable space $Z = [Z_1, \ldots, Z_Q]$ with Q observation vectors of dimension N. In our approach, the primitive is a closed contour. We build the multi-variable space with



(a) $\mathcal{L}_\sigma$ for $\sigma = 1$ (b) $\mathcal{L}_\sigma$ for $\sigma = 2$

(c) $\mathcal{L}_\sigma$ for $\sigma = 3$ (d) Set $\mathcal{L}$ for $\sigma = [1, 2, 3]$

Fig. 3: The image contours found at three different scales joined in the set $\mathcal{L}$

observations based on internal (geometrical features, e.g., circularity, roundness, area, perimeter) and external (e.g., mean gradient intensity, intensity inner area) properties of the contours.

Let $L_i \in \mathcal{L}$ be a contour, A_i its area and P_i its perimeter; we compute the circularity eq.(3) and the mean gradient intensity eq. (4) to build the multi-variable space $Z = [Z_1, Z_2]$.

$$Z_1 = \left[\frac{4\pi A_i}{P_i^2}, \;\; i = 0, \ldots, N \right]^T, \;\; N = card(\mathcal{L}) \tag{3}$$

$$Z_2 = \left[\frac{1}{P_i} \sum_{x \in L_i} \mid \nabla f(x) \mid, \;\; L_i \in \mathcal{L} \right]^T \tag{4}$$

RX Detector The RX anomaly detector [15] is commonly used to detect outliers on such data. The space Z models the set of contours $\mathcal{L}$ with $Q = 2$ feature vectors describing the circularity eq. (3) and the mean gradient intensity eq.(4). The RX detector gives an anomaly score to each contour taking into account the mean of the distribution and covariance between the Q-features through the Mahalanobis distance,

$$y_i = (z_i - \mu_Z)^T \Sigma_Z^{-1} (z_i - \mu_Z) \tag{5}$$

where $\mu_Z = [\mathrm{E}[z_1], \ldots, \mathrm{E}[z_N]]^T$ is the observations mean vector and Σ_Z^{-1} the $N \times Q$ covariance matrix of the data. If the data have normal random distribution,

then the score vector $Y = [y_i, \ldots, y_N]$ follows a chi-square distribution $\chi^2_Q(\varphi)$ with Q degrees of freedom, where φ is a confidence level [11]. The value of $\chi^2_Q(\varphi)$ with a confidence value $\varphi = 99.9\%$ operates as a threshold to identify all contours that behave as outliers in the multi-variable distribution. In our case, the contours belonging to a landing target appear as outliers in the vast majority of random contours belonging to the background.

With the previous strategy we preserve the anomalous contours having a value of mean gradient and circularity deviating from the principal mode of the distribution in the set $\widetilde{\mathcal{L}} = \{L_i \mid y_i > \chi^2_Q(\varphi)\}$. $\chi^2_Q(\varphi)$ is the value of the cumulative distribution at the confidence level φ and $\widetilde{\mathcal{L}} \subset \mathcal{L}$. It is essential to mention the importance of multi-scale contour detection of section 2.2; because it increases the number of samples in Z, allowing to build a richer multi-variable space.

In the set $\widetilde{\mathcal{L}}$ some contours make not part of a landing target. For example, in the figure 4, we can see that the paper sheet contours remain because they have a high value of circularity. The same occurs with the contours of those objects with an important value of mean gradient, as the number 4 at the top-left of the sheet or the rock textures of the background.

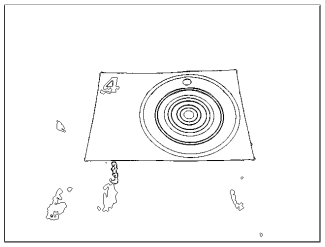
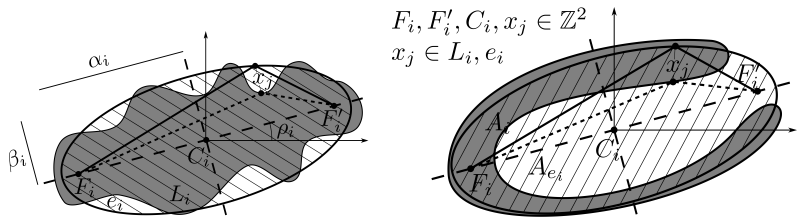


Fig. 4: The contours from Fig. 3d that behave as outliers in the multi-feature space Z with a confidence value of $\varphi = 99.9\%$

2.3 Gestalt Laws of Grouping

We use the Gestalt theory [17] to group the meaningful contours $L_i \in \widetilde{\mathcal{L}}$ and detect landing targets.

Goodness of Shape Since the landing targets have only circular contours, we evaluate the resemblance with an ellipse (to deal with the perspective deformation) of all contours. Considering an ellipse e_i that fits one gray contour L_i in Fig. 5a, we recover the centroid C_i, the rotational angle ρ, the semi-major axis α_i, the semi-minor axis β_i and the coordinates F_i and F'_i of the ellipse's foci. Then, the sum of the distances from any point of ellipse $x_j \in e_i$ to the foci is $\overline{x_j F_i} + \overline{x_j F'_i} = 2\alpha_i$. If the contour L_i is an ellipse, the value $d_i = \left|(\overline{x_j F_i} + \overline{x_j F'_i}) - 2\alpha_i\right|$ must be zero or negligible $\forall x_j \in L_i$.



(a) Affinity of a fit ω_i (b) Difference of area Δ_{A_i}

Fig. 5: Visual description of affinity of ellipse and difference of area

Based on the form of the landing target we estimate the the similarity using two measures,

$$\omega_i = \exp^{-\frac{d_i^2}{2\sigma^2}} \text{ the affinity of the fit and,} \tag{6}$$

$$\Delta_{A_i} = 1 - \frac{|A_{e_i} - A_i|}{\max(A_{e_i}, A_i)} \text{ the difference of area.} \tag{7}$$

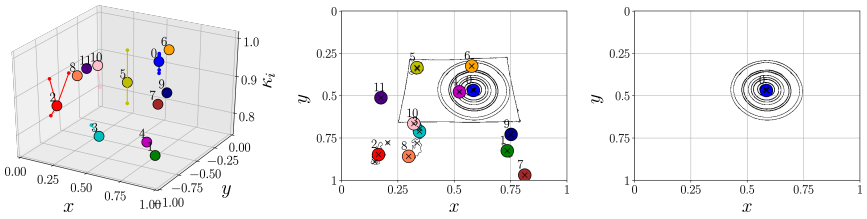
The affinity $\omega_i \to 1$ for contours closed to an ellipsoidal shape. However, if the contour L_i is a croissant shape (as in fig. 5b) then, the eq. (6) also has a high value (near to 1) but the contour is from being an ellipse. The variable in eq. (7) complements the affinity ω_i taking into account the area of the ellipse A_{e_i} and the area of the contour A_i. To calculate the similarity to an ellipse, we use the harmonic mean of both.

$$\kappa_i = \mathcal{H}(\omega_i, \Delta_{A_i}), \;\; \kappa_i \in (0, 1) \tag{8}$$

where $\kappa_i \to 1$ for contours ressembling to an ellipse and $\kappa_i \to 0$ otherwise. $\mathcal{H}$ denotes the harmonic mean $\mathcal{H} = N \left(\sum_{i=1}^{N} \xi_i^{-1} \right)^{-1}$.

Proximity Measure The Gestalt law of proximity states that we group those meaningful elements if they are spatially close to each other. In the case of contours, we take the coordinates of their centers C_i to measure their spatial proximity.

Affinity Clustering The normalized coordinates of the centroid C_i and the ellipse similarity κ_i map the contour $L_i \in \widetilde{\mathcal{L}}$ into the 3-D space $(0, 1) \in \mathbb{R}^3$. We use the affinity propagation clustering method [5] to group the contours using the matrix $X = [C_i, \kappa_i]$. This technique yields a set of clusters $\mathcal{C}_K \in \mathcal{C}(X)$. Because the landing target has ten different contours (see section 3), the clusters with $card(\mathcal{C}_K) \geq 10$ and an important similarity value $\mathcal{H}(\kappa_i) \geq 0.8$, represent the candidate contours of a landing target.



(a) Clusters obtained by affinity propagation

(b) Clusters projected on the image domain

(c) Target candidate cluster

Fig. 6: Clusters of contour from Fig. 4

The affinity propagation technique groups in $K = 12$ clusters the image contours from figure 4. In a 3D plot (fig. 6a), we see the influence of κ_i at clustering process. Projecting the clusters in a 2-D plane (fig. 6b), we notice that even if the contours are nearby, it can form a new cluster if there is a distant κ. A clear example is the clusters 0 and 4 (blue and purple, respectively) that correspond to the contour centers of the landing target and the center of the sheet of paper, they are close to each other but the similarity not. Applying the threshold values $card(\mathcal{C}_K) \geq 10$ and $\mathcal{H}(\kappa_i) \geq 0.8$ we obtain the candidate clusters to form a landing target (see fig. 6c).

Heretofore, we have built a model based on perceptual characteristics for the landing target detection. However, there could be false detections if there are circular objects with concentric borders in the image. We code an ID number in the target design to differentiate a landing target from an object with concentric circular edges. The coding of information allows discriminating between several landing targets and circular objects. The following section describes the landing target design as well as the coding and decoding technique.

3 Implementation

3.1 Landing target description

The landing target is formed by a set of black and white circles (see Fig. 7) that generate contours when stacked. Two of the circles ($\varnothing_9$ and $\varnothing_{10}$) have a constant diameter and form the ring that defines the target. The black circle ($\varnothing_{11}$) is an orientation reference and has the same diameter as the smallest circle, $\varnothing_{11} = \varnothing_1$. The other circles $\varnothing_1, \ldots, \varnothing_8$ are coding circles.

Landing Target ID Encoding Let $\text{Ø} = (\varnothing_1, \varnothing_2, \ldots, \varnothing_n)$ denote the nominal diameters of the coding circles. We can set the nominal diameters, e.g., $\varnothing_i = \frac{i}{n}\varnothing_n$ for a target without the encoding capability. To encode a number in the target form, we modify the nominal diameters Ø to obtain $\text{Ø}' = \left(\varnothing'_1, \varnothing'_2, \ldots, \varnothing'_n\right)$ by

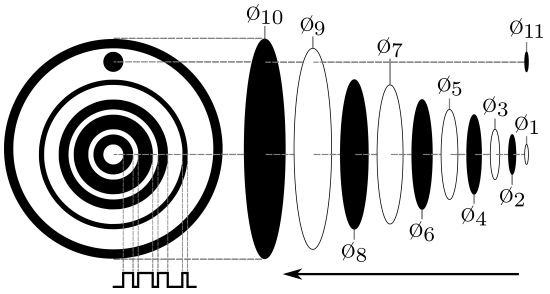


Fig. 7: Landing target design and description

adding/subtracting a positive constant Δh

$$\phi_i' = \begin{cases} \phi_i + \Delta h, & \text{if } w_i = 1 \\ \phi_i - \Delta h, & \text{otherwise} \end{cases} \tag{9}$$

and obtain a binary message $W = [w_1, \ldots, w_n]$. The message W is protected from errors by Hamming error-correction code [7]. It provides a set of different codewords $W = D \times M$ of size $n = k+m$, where D is useful data, $M = [I_k \mid 1-I_k]$ the generator matrix and I_k is the $k \times k$ the identity matrix. The data vector D comes from the decimal to binary conversion of the landing target ID* number. In our representation, we have experimented with $n = 8$ coding circles allowing to have four rings and $n = 8$ contours $\phi_1, \ldots, \phi_8$. This allows us to use the extended $[n, k]$ Hamming code with $k = 4$ data bits and $m = 4$ parity bits to generate $2^4 = 16$ landing targets.

Landing Target ID Decoding After the clustering stage of section 2.3, we rank by size the ellipses' major axes α_i by size and normalize them w.r.t. the largest value α_{10} to obtain $\boldsymbol{\alpha} = \frac{\phi_{10}}{\alpha_{10}}(\alpha_1, \ldots, \alpha_{10})$

We compare the received and normalized axis $\boldsymbol{\alpha}$ with the nominal diameters of the coding circles Ø and transform them into a binary vector $\widehat{W}$;

$$\widehat{W} = \begin{cases} 1, & \text{if } \alpha_i - \phi_i > 0 \\ 0, & \text{otherwise} \end{cases} \quad \forall i = 1, \ldots, n \tag{10}$$

The Hamming syndrome vector $S = \widehat{W} \times H^T$ (with $H = [1 - I_k \mid I_k]$ as the parity-check matrix) indicates whether an error has occurred. The syndrome is a null vector $S = 0$ when no error has occurred, otherwise, $S \neq 0$ and $\widehat{W} = W + E$. The element $e_i = 0$ of the error vector $E = H^T - S$ indicates an error at the position i. The $[8, 4]$ Hamming code can find up to two erroneous bits and correct one. Once the algorithm corrects the error (if there is), the vector $\widehat{W}$ is decoded by using the modulo 2 of the product $\widehat{D} = \widehat{W} \times M^T$.

*Identification number

3.2 Validation and tests

The presented strategy was validated on landing target images under simulated and real situations. We tested the algorithm in a synthetic image database which simulates four image degradations: noise, shadows, target deformation and change of size. For the real situations, we carried out several tests in indoor and outdoor scenarios. Figure 8 shows three interesting experiments and the output image of each stage of section 2.

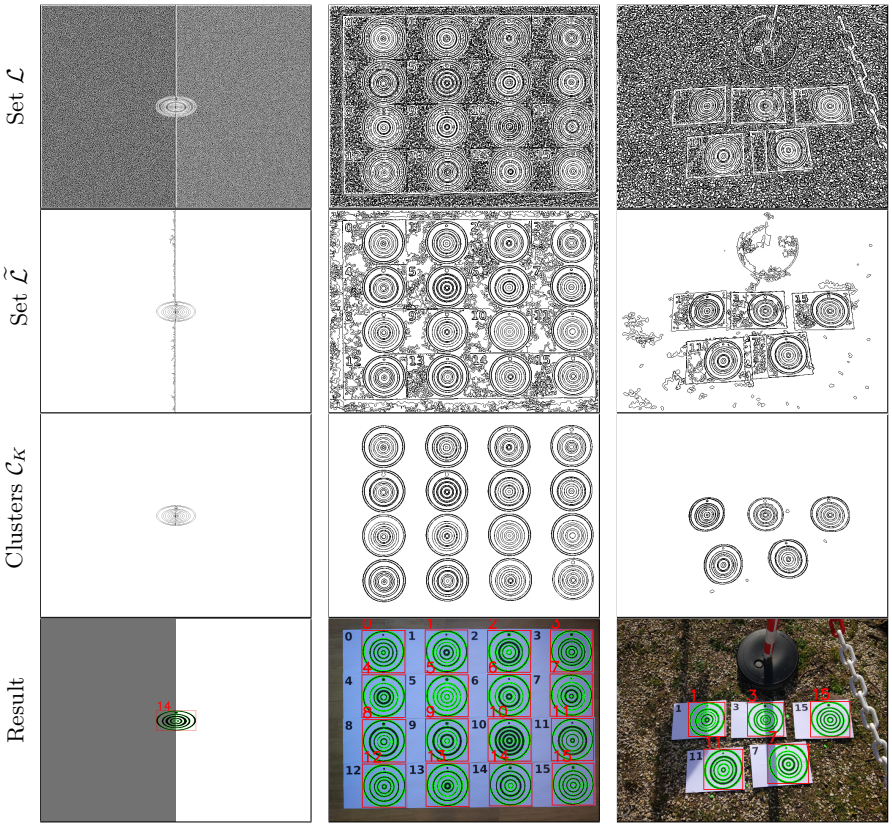


Fig. 8: Algorithm validation: (a) Target under simulated degradations, (b) The 16 targets in an indoor environment, (c) Five targets in an outdoor scenario under non-controlled image degradations

The first experiment (Fig. 8a) shows the four synthetic degradations together on landing target ID 14. In this context, the synthetic image represents the values of degradation maximum that the algorithm supports. Second experiment (Fig.

8b) was done in an indoor space to show the sixteen possible landing targets. In the scene there are no other objects. Finally, the last experiment (Fig. 8c) shows five landing targets in a more complex outdoor environment. Notice the presence of other objects, different background textures, irregular shadows and perspective deformation and change of scale of the landing targets.

In the three experiments, i) the non-accidentalness estimation stage eliminates the contours generated by noise with low circularity and mean gradient values; ii) the grouping stage filters random contours generated by intensity changes like shadows to keep contours with an important value of similarity and proximity. The compilation of the experiments carried out under real conditions can be seen in https://youtu.be/igsQc7VEF2c.

4 Conclusion and Future Extensions

We have described the procedure for the landing target detection and recognition based on a perception model. The algorithm is based on the Helmholtz non-accidentalness principle and the Gestalt theory. The non-accidentalness estimation is performed in a multi-feature object space built from the image contours at different scales. This approach allows us to obtain scene information avoiding the loss of information because of the objects' change of size or the presence of shadows and noise. We have used the similarity and proximity Gestalt laws to group the contours and build a perceptual object and the Hamming error codes to perform the landing target recognition. The experiments show that the proposed methodology for the detection of landing targets is robust to uncontrolled light conditions and other images degradations existing in complex environments.

For this particular work, the objective was to detect a landing target, a circular object (basic geometric shape) with concentric borders; however, the presented methodology has a wide extension capacity. At the moment we use only the image contours to measure the circularity and the mean intensity gradient. A future extension could include the use of other image primitives, such as points, regions, texture or color, and other object features. Similarly with the grouping laws, we explored only the proximity and similarity laws; however, it is possible to use other laws such as alignment, symmetry or continuity. These ideas will allow creating new descriptive feature spaces of scenes and objects in an image; with the possibility of make combinations between grouping laws, features and image primitives to detect some particular object.

Finally, our contribution proposes a perception model that utilizes the a contrario theory but avoids the mathematical complexity of setting the NFA to adjust threshold values or parameters in specific situations.

Acknowledgments

This research is partially supported by the Mexican National Council for Science and Technology (CONACYT).

References

1. Araar, O., Aouf, N., Vitanov, I.: Vision Based Autonomous Landing of Multirotor UAV on Moving Platform. Journal of Intelligent & Robotic Systems 85(2), 369–384 (Feb 2017)
2. Attneave, F.: Some informational aspects of visual perception. Psychological Review 61(3), 183–193 (1954)
3. Carrio, A., Sampedro, C., Rodriguez-Ramos, A., Cervera, P.C.: A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. J. Sensors 2017, 3296874:1–3296874:13 (2017)
4. Desolneux, A., Moisan, L., Morel, J.M.: From Gestalt Theory to Image Analysis: A Probabilistic Approach. Interdisciplinary Applied Mathematics, Springer-Verlag, New York (2008)
5. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. Science (New York, N.Y.) 315(5814), 972–976 (Feb 2017)
6. Furukawa, H.: Deep Learning for End-to-End Automatic Target Recognition from Synthetic Aperture Radar Imagery. arXiv:1801.08558 [cs] (Jan 2018)
7. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal 29(2), 147–160 (Apr 1950)
8. Lacroix, S., Caballero, F.: Autonomous detection of safe landing areas for an UAV from monocular images. In: In IEEE/RSJ International Conference on Intelligent Robots and Systems (2006)
9. Lange, S., Sünderhauf, N., Protzel, P.: Autonomous Landing for a Multirotor UAV Using Vision. In: In SIMPAR 2008 Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots. pp. 482–491 (2008)
10. Lee, J., Wang, J., Crandall, D., Šabanović, S., Fox, G.: Real-Time, Cloud-Based Object Detection for Unmanned Aerial Vehicles. In: 2017 First IEEE International Conference on Robotic Computing (IRC). pp. 36–43 (Apr 2017)
11. Lu, C.T., Chen, D., Kou, Y.: Multivariate spatial outlier detection. International Journal on Artificial Intelligence Tools 13(04), 801–811 (Dec 2004)
12. Marr, D., Hildreth, E.: Theory of edge detection. Proc. R. Soc. Lond. B 207(1167), 187–217 (Feb 1980)
13. Mumford, D., Shah, J.: Optimal approximations by piecewise smooth functions and associated variational problems. Communications on Pure and Applied Mathematics 42(5), 577–685 (Jul 1989)
14. Petitot, J.: Neurogéométrie de la vision: modèles mathématiques et physiques des architectures fonctionnelles. Editions Ecole Polytechnique (2008)
15. Reed, I.S., Yu, X.: Adaptive multiple-band CFAR detection of an optical pattern with unknown spectral distribution. IEEE Transactions on Acoustics, Speech, and Signal Processing 38(10), 1760–1770 (Oct 1990)
16. Sezgin, M., Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. J. Electronic Imaging 13(1), 146–168 (Jul 2010)
17. Wertheimer, M.: FormsUntersuchungen zur Lehre von der Gestalt II. Psycologische Forschung 4, 301–350 (1923)
18. Witkin, A.: Scale-space filtering: A new approach to multi-scale description. In: ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing. vol. 9, pp. 150–153 (Mar 1984)
19. Yao, H., Yu, Q., Xing, X., He, F., Ma, J.: Deep-learning-based moving target detection for unmanned air vehicles. In: 2017 36th Chinese Control Conference (CCC). pp. 11459–11463 (Jul 2017)

Parallel Algorithm for Concurrent Computation of Connected Component Tree

P. Matas[1,2], E. Dokladalova[1], M. Akil[1], T. Grandpierre[1],
L. Najman[1], M. Poupa[2], and V. Georgiev[2]

[1] IGM, Unité Mixte CNRS-UMLV-ESIEE UMR8049, Université Paris-Est,
Cité Descartes, BP99, 93162 Noisy le Grand, France
{matasp, e.dokladalova, akilm, grandpit, l.najman}@esiee.fr
[2] Department of Applied Electronics and Telecommunications, University of West
Bohemia, Univerzitní 26, 306 14 Plzeň, Czech Republic
{pmatas, poupa, georg}@kae.zcu.cz

Abstract. The paper proposes a new parallel connected-component-tree construction algorithm based on line independent building and progressive merging of partial 1-D trees. Two parallelization strategies were developed: the parallelism maximization strategy, which balances the workload of the processes, and the communication minimization strategy, which minimizes communication among the processes. The new algorithm is able to process any pixel data type, thanks to not using a hierarchical queue. The algorithm needs only the input and output buffers and a small stack. A speedup of 3.57 compared to the sequential algorithm was obtained on Opteron 4-core shared memory ccNUMA architecture. Performance comparison with existing state of the art is also discussed.

1 Introduction

Computer vision systems are asked to furnish high performance and be flexible for a large variety of existing or possible applications. One of the global problems of the vision system design is how to achieve these two characteristics simultaneously. If the high performance is achieved by an optimization effort which means a kind of system specialization, it will (by definition) limit its flexibility.

The connected component tree (CCT) based image processing algorithms seem to be very promising from this point of view. They allow bridging the gap between low- and high-level processing implementations. They have been used for filtering [1, 5] as well as the image analysis: motion extraction [1], watershed segmentation [6, 2, 14], segmentation of astronomical images [3] or data visualization [13].

Fig. 1 shows typical stages of an application based on CCT. We can see the advantage of these methods: once the CCT is constructed, the processing is performed on the tree by graph transformation(s), and only one data structure is used from low-level to high-level processing. In addition, the graph transformations are applicable to any dimension (1D, 2D, 3D …).

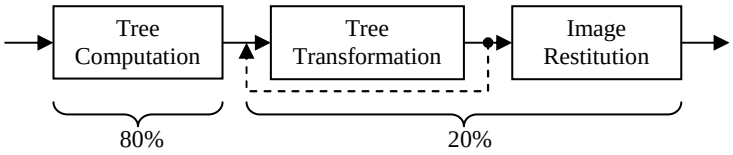


Fig. 1 Stages of a typical CCT-based application and times of execution

On the other hand, the main bottleneck is represented by the CCT construction, consuming about 80% of the application execution time (see Fig. 1), which is penalizing for a lot of practical applications.

In the past, several algorithms have been proposed in order to solve this problem. In majority of cases they remain sequential and the improvement relies on fast data structures (FIFO-like) [1] or on optimization of computational complexity [2]. Some parallelization effort has been done on shared-memory computers by [10, 5]. However, if the obtained performances are interesting they are insufficient for real-time and not adapted to the embedded systems [8].

In this paper, we present new parallel algorithm for computation of the CCT. The algorithm proceeds line by line (inspired by [4]). Then the line trees are progressively merged. Since the line-based CCT construction is extremely fast, a new parallelization strategy is needed for concurrent implementation. This paper presents and evaluates two parallelization strategies: i) parallelism maximization, ii) limited communication among computing blocks. In addition, our algorithm makes use of memory-aware data structure hence it is more suitable for embedded system implementation.

The paper is organized as follows. Section 2 shortly discusses existing sequential and parallel algorithms and analyses their computational complexity. In Section 3, we give basic mathematical background. Section 4 presents the new proposed algorithm, the parallelization approaches and their theoretical evaluation. Finally, some experimental results on real multi-processor systems are presented in Section 5.

2 State of the art

Three main classes of sequential algorithms exist in the literature:

- *Flooding-based algorithms* [1]: processing starts from the image pixel having the lowest level. A depth-first traversal of tree components, similar to flood-fill, is performed. In general, the flooding process relies on the use of ordered data structures: i) hierarchical queues, unusable for floating-point pixel representation, ii) priority queues, inefficient from the time and memory point of view.
- *Emerging-based algorithms*: image pixels are processed in decreasing order of luminosity. It requires prior pixel sorting which could be done efficiently. The emerging components are processed as disjoint sets of pixels, based on Tarjan's Union-Find algorithm [7]. In [2], both total path compression and weighting are used to speedup the disjoint set algorithm and the algorithm complexity is quasilinear. [3, 8, 9] use only total path compression in order to save memory.

- *1-D algorithms*: it is a special category where the CCT is built on 1-D signal. Thus, the pixel processing ordering is unnecessary and the tree can be built in linear time [4]. These algorithms are extremely fast, but they cannot process 2-D data. However, if tree merging is added [10], they are usable for any dimension, so we have chosen this approach.

In order to accelerate the execution time, Meijster [10] studies the first implementation of CCT computation on shared memory machines. Recently, Wilkinson [5] has published a modification of the Meijster's approach and has demonstrated the computation performance on 3-D data. The principle consists of image division into regular domains. A modification of Salembier's algorithm [1] is used to build a CCT of each domain independently. Then, the trees of the domains are merged in a binary-tree fashion. See **Table 1** which summarizes the complexity analysis of the existing CCT computation algorithms (sequential and parallel).

Table 1. Complexity analysis. N is the total number of pixels in the image, G is the number of grey levels of the image, α is a very slow-growing "diagonal inverse" of the Ackermann's function, $\alpha(10^{80}) \approx 4$. For the Wilkinson's algorithm, P is the number of processes

	Time complexity	**Memory requirements calculation hints**	**Data types**
Salembier [1]	$O(NG)$	$4N + 3G$ + stack	small int
Najman-Couprie [2]	$O(N\,\alpha(N))$	$7N + G$ + stack	int/float
Berger [3]	$O(N \log N)$	$4N$ + stack	int/float
Levillain [12]	$O(NG \log N)$	$2N$ + stack	int/float
Menotti (1D) [4]	$O(N)$	$3N + G$ + stack	int/float
Wilkinson (3D) [5] (building + merging)	$O(NG/P + N^{2/3}G \log N \log P)$	$3N + 3G$ + stack	small int

3 Mathematical background

Let us consider a function $f: \mathbf{Z}^2 \rightarrow \mathbf{R}$ an image associated with 4-connectivity (can be generalized to any dimension and connectivity), where below $V = \text{supp}(f)$ denotes the set of points (pixel coordinates) of the image and $\mathbf{R}$ is the real number set. We call a *k-level connected component* ($C \subset V$) if all of the following conditions are met:

1. C is connected with regard to the 4-connectivity.
2. $\forall\, x \in C:\ f(x) \geq k$
3. No other point $y \in V$ can be added to C without violating the conditions 1 or 2.

We call $h(C) = \min\{f(x)\ ;\ x \in C\}$ the *altitude* of component C. For each point $x \in V$ we define $C_f(x)$ as the component of the image f, which has the altitude $h(C_f(x)) = f(x)$ and which contains the point x.

The connected components of the image may be organized, thanks to the inclusion relation, to form a rooted tree structure called ***connected component tree***: For each

two components C_1, C_2 of the image f, we say that C_1 is the parent of C_2 precisely if $C_2 \subset C_1$ and there is no other component C_3 of image f, such that $C_2 \subset C_3 \subset C_1$.

We call the subset $C' = \{ x \in C ; f(x) = h(C) \}$ of component C as the *core* of the component C. It is the set of points of the component C, which belong to no descendant of C. Note that each point of V belongs to exactly one component core.

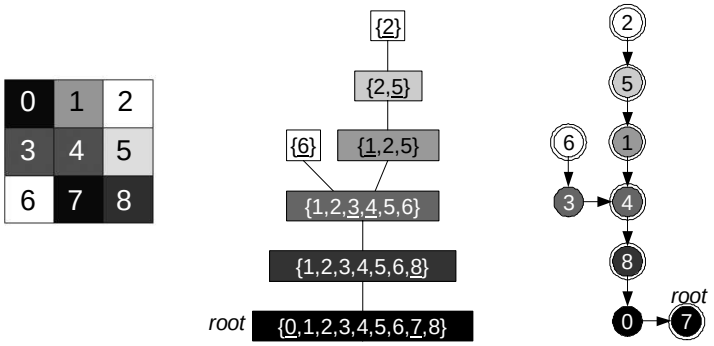


Fig. 2 Component trees**:** a) an example image f with points numbered in scan-line order; white background represents the highest level of f. b) the component tree of the image f; the points of the component's core are underlined. c) the point-tree of the image f ; level roots are marked with double circles

In our algorithm we use a memory-aware representation [10], [5] of the component tree called ***point-tree*** [11]. The point-tree (PT) of the image f is a rooted tree, whose nodes are points of the image f and where points of each component of the image f form a subtree of PT. Note that there may be more than one valid point-tree corresponding to a given image.

The edges of PT are represented by parent pointers stored in an array named $par : V \rightarrow V \cup \{\perp\}$ where $\perp$ stands for null pointer. For each point $x \in V$, $par[x]$ is the parent of x if x is not the root of PT and $par[x] = \perp$ if x is the root of PT. We define $f(\perp) = -\infty$.

It can be shown that for each point $x \in V$:

- $f(par[x]) \leq f(x)$
- If $f(par[x]) < f(x)$ then $C_f(par[x])$ is the parent component of $C_f(x)$ and x is the root of PT's subtree composed of points of $C_f(x)$. We say that x is the level root of the component $C_f(x)$ and of all its core's points.
- If $f(par[x]) = f(x)$ then $C_f(par[x]) = C_f(x)$ and we say that x is not a level root.

4 New algorithm description

The algorithm proceeds in two steps: a partial point tree is computed independently for each line (Algorithm 1) and then the partial trees of neighboring lines have to be merged together.

Partial point-tree computation

Menotti's 1-D algorithm was modified to produce a PT, which is suitable for merging and parallelization, instead of a component tree structure and component mapping. The algorithm processes the points of the line in a single linear scan from left to right and stores the result as the point-tree to allow a subsequent merging. The algorithm uses only a stack (LIFO) to store the points, whose parent pointers could not be determined yet. The stack supports these operations:

- **`StackPush(`*x*`)`** : Adds the point *x* to the top of the stack.
- **`StackPop()`** : Removes one point from the top of the stack.
- **`StackLast()`** : Returns the point at the top of the stack without stack modification or $\perp$ if the stack is empty.
- **`StackEmpty()`** : Returns true if the stack is empty.

The first point of the line is a level root, because it is surely the leftmost point of some component core. Variable *r* is initialized to this first point and the scan starts from the second point of the line. Note that the leftmost point of each component core is treated as a level root.

The following invariant holds during the scan: before and after each iteration of the scan, i) the variable *r* holds the level root of the last processed point and ii) the stack contains all level root ancestors of *r* encountered so far, ordered by their levels, the highest level on the top of the stack.

The parent pointer of each point is assigned exactly once, just before the point is dropped from the stack; from the variable *r* and the variable *p* used during the scan. The parent pointer is always set to point to a level root, so a perfectly compressed point-tree is produced.

A new point *p* is processed in each iteration of the scan. Its level $f(p)$ is compared to the level $f(r)$. There are three possibilities:

- $f(r) < f(p)$: point *p* is the leftmost point of a new component core, so it is a level root. Point *r* is a (possibly indirect) ancestor of *p*, because *r* is the level root of its left neighbor. Current *r* is pushed to the stack and *p* is set as the new value of *r*. No point is dropped, so no parent pointer is set.
- $f(r) = f(p)$: point *p* belongs to the same component core as the last processed point and *r* is its level root, so *r* is assigned to $par[p]$ and *p* is dropped.
- $f(r) > f(p)$: the component represented by point *r* is completed and its parent has to be determined. Let *q* be the point on the top of the stack. Either *p* or *q* is the parent of *r*, depending on their levels, so $f(p)$ is compared to $f(q)$. Again, there are three possibilities:
  - The stack is empty or $f(q) < f(p)$: point *p* is the leftmost point of a new component core, so it is a level root. It is also the parent of *r*, so *p* is assigned to $par[r]$ and *r* is dropped. Point *p* is set as the new value of *r*.
  - $f(q) = f(p)$: the points *q* and *p* belong to the same component core. The point *q* is its level root and the parent of *r*, so *q* is assigned to both $par[p]$ and $par[r]$ and both *p* and *r* are dropped. Point *q* is removed from the stack and set as the new value of *r*.
  - $f(q) > f(p)$: point *q* is the parent of *r*, so it is assigned to $par[r]$ and *r* is dropped. Point *q* is removed from the stack and set as the new value of *r*. But

now, $f(r)$ is still greater than $f(p)$. This means that the component represented by the new value of r is completed and its parent has to be determined now. This is done by repeating the decision process with the same p, the new value of r and the new state of the stack.

After the scan finishes, r contains the level root of the last point of the line and all its ancestors are in the stack. They are removed from the stack one by one and parent pointers are set accordingly. The last point removed from the stack is the tree root.

Algorithm 1 1-D algorithm for computation of point-tree

$V = \{0 \dots W-1\} \times \{0 \dots H-1\}$
$line \in \{0 \dots H-1\}$ is the number of the line to be processed
$V_{line} = \{0 \dots W-1\} \times \{line\} = \{ (i, j) \in V ; j = line \}$ is the set of points of one line
Input: image $f : V_{line} \rightarrow \mathbf{R}$
Output: point-tree $par : V_{line} \rightarrow V_{line} \cup \{\perp\}$

```
procedure build1D(line : integer) =
 1     var r : point := (0, line) ;
 2     for p : point := (1, line) ... (W - 1, line) do
 3          if f(r) < f(p) then
 4               StackPush(r) ;
 5               r := p ;
 6          elsif f(r) = f(p) then
 7               par[p] := r ;
 8          else loop
 9                    var q : point := StackLast() ;
10                    if f(q) < f(p) then
11                         par[r] := p ; r := p ;
12                         break ;
13                    elsif f(q) = f(p) then
14                         par[r] := q ; par[p] := q ; r := q ;
15                         StackPop() ; break ;
16                    else
17                         par[r] := q ; r := q ;
18                         StackPop() ;
19     end; end; end; end;
20     while StackEmpty() = false do
21          par[r] := StackLast() ; r := StackLast() ;
22          StackPop() ;
23     end;
24     par[r] := ⊥ ;  (* r is root *)
end;
```

Merging of partial point-trees

Generally speaking, we take two adjacent point-trees as input and modify their parent pointers to create a single point-tree. The merging operation is done in procedure `connect`(x, y), which is executed for each pair of points x and y, where x is in the first point-tree, y is in the second point-tree and x and y are neighbors in 4-connectivity sense. Procedure `connect` follows the parent pointer paths from x and y respectively to the root of the tree and changes the parent pointers to form a single path. The new path includes nodes visited along both paths in correct order of levels. When the two parent pointer paths meet, the procedure is terminated. Note that the principle is the same as used in [5], with some simplifications.

Algorithm 2 Merging process of two adjacent point-trees

The first point-tree starts at line a and ends at line $border - 1$, the second point-tree starts at line $border$ and ends at line b ($0 \le a < border \le b < H$)
$V_{a,b} = \{0 \dots W-1\} \times \{a \dots b\} = \{ (i, j) \in V ; a \le j \le b \}$ is the set of points of the two trees
Input: image $f : V_{a,b} \to \mathbf{R}$; point-tree $par : V_{a,b} \to V_{a,b} \cup \{\perp\}$
Output: point-tree $par : V_{a,b} \to V_{a,b} \cup \{\perp\}$

```
 1    procedure levroot(x : point) returns point =
 2         if f(x) = f(par[x]) then
 3              par[x] := levroot(par[x]) ; return par[x] ;
 4         else
 5              return x ;
 6    end; end;

 7    procedure connect(x, y : point) =
 8         x := levroot(x) ;  y := levroot(y) ;
 9         if f(y) > f(x) then swap(x, y) ; end;
10         while x ≠ y do
11              if par[x] = ⊥ then
12                   par[x] := y ;  x := y ;
13              else
14                   var z := levroot(par[x]) ;
15                   if f(z) > f(y) then
16                        x := z ;
17                   else
18                        par[x] := y ; x := y ; y := z ;
19    end; end; end; end;

20    procedure merge(border : integer) =
21         for all neighbors x, y between lines (border – 1) and border do
22              connect(x, y) ;
23    end; end;
```

4.1 Concurrent implementation

Parallelism Maximization Strategy. The point-trees of individual lines are independent after the building; there are no parent pointer links between the nodes of different lines. To create a complete point-tree of the image, which consists of H lines, each two adjacent lines have to be merged. When two lines merging started, two originally independent point-trees become connected and a larger point-tree is formed. No other process should work on the two trees being merged. To achieve maximal parallelism, it is the best to merge point-trees of the same or similar sizes, whenever possible. For this reason, each merge step of our algorithm merges two point-trees, which consist of 2^k lines each, into one point-tree of 2^{k+1} lines. Only if the height H of the image is not a power of two, then the point-tree, which includes the last line of the image, may be smaller than the point-tree which it is merged with.

The algorithm consists of H building operations and $(H - 1)$ merging operations. Their dependencies are shown in figure 3. Each building operation accepts one line of the image, i.e. $W \times 1$ pixels, n bits per pixel as the input and produces the point-tree of the same size, m bits per parent pointer.

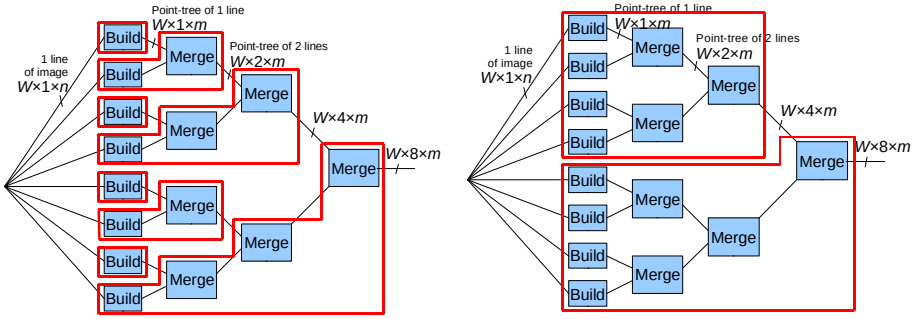


Fig. 3. Data dependency graph. a) Parallelism maximization strategy, b) Communication minimization strategy. The enclosing polygons show which operations are done by a single process

The algorithm is executed in $P \in \mathbf{N}$ (natural number) independent parallel processes (threads). Numbered lines in a queue are fetched by the individual processes. The queue is represented by a shared *next_line* counter. The counter has to be read first and then incremented on each fetch. This is done by the synchronized procedure `get_next_line`, where "Synchronized" means that if two processes attempt to enter the procedure at the same time, one of them has to wait until the other leaves the synchronized procedure. When a line is fetched by the process, the process performs the building operation. Then the process proceeds to the cascade of merging operations following the path from the build job to the right in the data dependency graph. The process performs as many merging operations as possible without waiting. Each merging job requires two inputs (two point-trees). One of the inputs is carried by the process which is about to perform the merging job. The other input has to be supplied by another process. The boolean array *block_ready* is used to keep track of merging jobs which are ready to be done. An element of *block_ready* is true if the corresponding merging job has at least one input ready. The process, which attempts the merging, calls the synchronized procedure `test_merge`. This procedure reads the previous value of the corresponding element of *block_ready*, which it will return, and sets it true. If the read value is true (the process is the second one to attempt this merging job), it means that the other input of the merging job is ready and the job is performed by the process. If the read value is false (the process is the first one to attempt the merging job), the other input is not ready yet, so the process leaves the cascade of merging operations and fetches next unassigned line to be built. We can see that each building job will be done exactly once, because each of them is fetched by one process. Also each merging job will be done exactly once, because exactly one process attempts each merging job as the second. If a process attempts to fetch next unassigned line, but there are no more lines left, the process exits. The algorithm ends when all processes exit.

Algorithm 3 Concurrent implementation

Input: image $f : V \rightarrow \mathbf{R}$
Output: point-tree $par : V \rightarrow V \cup \{\perp\}$

```
 1    var next_line : integer := 0 ;
 2    var block_ready[1..H] : boolean ; (elements initialized to false)
 3    synchronized procedure get_next_line() returns integer =
 4         if next_line < H then
 5              return next_line++ ;
 6         else
 7              return -1 ;
 8    end; end;

 9    synchronized procedure test_merge(border : integer)
      returns boolean =
10         var result : boolean := block_ready[border] ;
11         block_ready[border] := true ;
12         return result;
13    end;

14    process line_parallel_tree() =
15         while (var line := get_next_line()) != -1 do
16              build1D(line) ; (* Build partial tree*)
17              var block_no := line ;
18              var block_size := 1 ;
19              while block_size < H do (*Merge as deep as possible*)
20                   block_no := block_no div 2 ;
21                   block_size := block_size * 2 ;
22                   var border := (block_no + 1/2) * block_size;
23                   if border < H then
24                        if test_merge(border) = true then
25                             merge(border) ;
26                        else
27                             break ;
28    end; end; end; end; end;
```

Communication Minimization Strategy. The algorithm is executed in $P \in \mathbf{N}$ (natural number) independent parallel processes (threads). The entire image is divided into P roughly equal-sized blocks of approximately (H / P) lines (the numbers of lines per block have to be integers). The data dependency graph is the same as for the line parallel algorithm only if the block size is a power of 2 lines (Fig. 3b). If the block size is not a power of 2 then the data dependency graph is slightly different. Each block is assigned to one process. The process builds and merges all lines of the block into the point-tree of the block, just like the line parallel algorithm running in a single process does. It means that after building of a line, as many merging steps as possible are done. Synchronization is no more necessary, because the job execution order is known in advance. After the block was built and merged, it is merged with the other blocks. If we consider processing of the block as a single building operation, the data dependency graph applies to the merging of the blocks as for the line parallel algorithm. One of the processes, which reach a merging operation, terminates and the other process proceeds with merging. The final point-tree is done when last two blocks are merged.

4.2 Time complexity and memory requirements

The time complexity of the building operation (procedure `build1D`) was already analyzed in [4] and equals $O(W)$.

For its memory requirements, let us consider that: no point can be inserted twice into the stack; no two points simultaneously present in the stack can have the same level; the last point is never inserted into the stack; the points with the highest level are never inserted into the stack. Thus, the maximum stack size is $\min(W-1, G-1)$.

The merging operation time complexity was already analyzed by Wilkinson [5] and is $O(WG \log N)$. The merging operation does not need any additional memory except input and output buffers. The recursive procedure `levroot` needs a stack to store the points for path compression. It is possible to avoid the need for stack by implementing this procedure without recursion, but the parent pointers have to be read twice.

The total time complexity of the algorithm is dominated by the merging operations and is $O(NG \log N)$ for the whole image with N pixels and G grey levels.

The total memory requirements are input buffer of N pixels, output buffer of N memory pointers and a stack of size $\min(W-1, G-1)$.

5 Experimental results

The algorithm described in the previous section was implemented in C and compiled using Visual C++ 2005 Express Edition. The benchmarks were run on two different computers with 2 and 4 CPU cores:

- 2-core machine: Portable computer with Intel Pentium Dual-Core T2330 @ 1.6 GHz, 1 GB RAM, Windows XP Professional SP2
- 4-core machine: 2× dual-core AMD Opteron 280 @ 2.4 GHz, 16 GB ccNUMA RAM, Windows Server 2003 R2

We measured the wall clock times (real time elapsed from start to end of the task) of the two parallelization strategies (parallelism maximization and communication minimization) for 1, 2, 4, 8, 16 threads on a test image of size 784×576 px (8 bits).

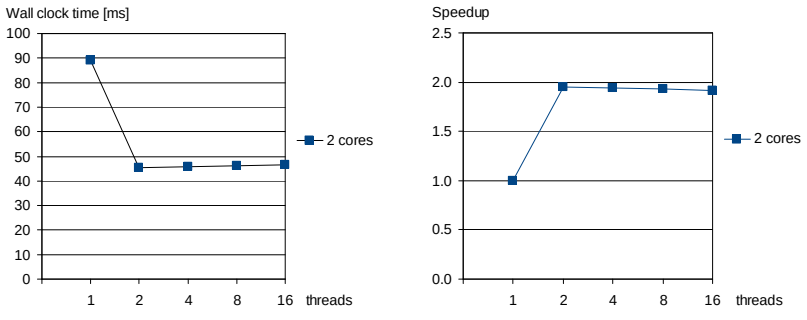


Fig. 4. Parallelism maximization strategy timings: a) Wall clock time versus number of threads, b) Speedup with respect to the sequential algorithm versus number of threads.

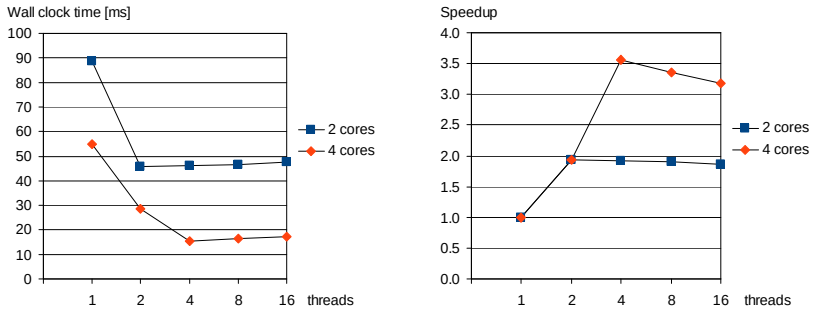


Fig. 5. Communication minimization strategy timings: a) Wall clock time versus number of threads, b) Speedup with respect to the sequential algorithm versus number of threads.

On the 2-core machine, we get maximal speedup 1.95 from 89.0 ms to 45.6 ms for 2 threads for the parallelism maximization strategy and the results for the communication minimization strategy are very similar. On the 4-core machine, we get maximal speedup 3.57 from 55.1 ms (0.122 μs/pixel) to 15.4 ms (0.034 μs/pixel) for 4 threads for the communication minimization strategy. Maximum speedup is obtained when number of cores and threads match. Theoretical maximal speedup is 2.0 and 4.0 for 2 and 4 cores respectively. The numbers measured are less than theoretical because of the workload imbalance and overhead.

The time increases roughly linearly with increasing width of the image.

Results obtained by Wilkinson on the 2× dual-core Opteron machine with 8 GB RAM operating on a 512^3 8-bit volumetric data set are as follows: Wall clock time for 1 thread was 73 s (0.54 μs/voxel), wall clock time for 64 threads was 13.7 s (0.102 μs/voxel), that gives the speedup 5.3.

6 Conclusions

We have designed a new parallel algorithm for CCT construction. The algorithm design was focused on maximal parallelism. Both parallelization strategies minimize the traffic between the execution units and the memory. The communication minimization strategy also minimizes the traffic among the execution units, but its disadvantage is the fixed process workload assignment. The parallelism maximization strategy has advantage when the data stream is of serial form from the sensor, because it takes the lines of the image from top to bottom, one by one. The parallelism maximization strategy provides a usable workload balancing, but depends on a significant amount of synchronization among processes.

The Salembier-based Wilkinson's algorithm is slow when the difference of neighbor levels is large. The new algorithm does not depend on absolute levels, so it is able to process any pixel data type (including floating point representation) without large performance loss, thanks to not using a hierarchical queue.

Wilkinson obtains the biggest speedup for number of threads far higher than number of physical CPUs. He explains that by improvement of data locality due to

dividing of the image into smaller blocks. This does not apply to our algorithm, because it proceeds by lines of size independent of number of threads, so best speedup is obtained when number of threads equals number of physical cores. Additional increase of thread count extends the time due to thread management overhead.

The new algorithm takes approximately 3 times less time to process one image element compared to Wilkinson's algorithm.

In the future we will evaluate a tradeoff between the presented strategies and optimize it for the Cell platform, a powerful multiprocessor on a chip.

References

1. P. Salembier, A. Oliveras, and L. Garrido. Anti-extensive connected operators for image and sequence processing. *IEEE Trans. on Image Proc.*, 7(4):555–570, April 1998
2. L. Najman and M. Couprie. Building the component tree in quasi-linear time. *IEEE Transactions on Image Processing*, 15/11: 3531–3539, 2006
3. C. Berger, Th. Géraud, R. Levillain, N. Widynski, A. Baillard, E. Bertin. Effective component tree computation with application to pattern recognition in astronomical imaging. ICIP 2007
4. D. Menotti, L. Najman and A. de Albuquerque Araújo. 1D Component Tree in Linear Time and Space and its Application to Gray-Level Image Multithresholding. *Proceedings of the 8th International Symposium on Mathematical Morphology*, Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 437–448
5. M. H. F. Wilkinson, Hui Gao, W. H. Hesselink, Jan-Eppo Jonker and Arnold Meijster. Concurrent Computation of Attribute Filters on Shared Memory Parallel Machines. Submitted for *Transactions on Pattern Analysis and Machine Intelligence*, 2007
6. M. Couprie, L. Najman and G. Bertrand. Quasi-linear algorithms for the topological watershed. *Journal of Mathematical Imaging and Vision*, Volume 22, Issue 2 - 3, May 2005, Pages 231 - 249
7. R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22: 215–225, 1975
8. N. Ngan, F. Contou-Carrère, B. Marcon, S. Guérin, E. Dokládalová, M. Akil. Efficient hardware implementation of connected component tree algorithm. Workshop on Design and Architectures for Signal and Image Processing, DASIP 2007, Grenoble, France.
9. C. Berger, N. Widynsky. Using connected operators to manipulate image components. *Report, LRDE Seminar*, July 2005
10. A. Meijster. Efficient Sequential and Parallel Algorithms for Morphological Image Processing. PhD thesis, Rijksuniversiteit Groningen.
11. Berger, Ch.; Geraud, T.; Levillain, R.; Widynski, N.; Baillard, A.; Bertin, E. Image Processing, 2007. ICIP 2007. IEEE International Conference on Volume 4, Issue , Sept. 16 2007-Oct. 19 2007 Page(s):IV - 41 - IV – 44
12. B. Deloison. Recherche et développement en traitement d'image: Utilisation de l'arbre des composantes pour la fusion d'images. Report from graduate project, ESIEE Paris, June 2007.
13. Chiang, Y.-J., Lenz, T., Lu, X., and Rote, G., Simple and optimal output sensitive construction of contour trees using monotone paths. *Comp.Geometry: Theory and Applications*, 30(2):165–195, 2005.
14. Mattes, J., Demongeot, J., Efficient algorithms to implement the confinement tree. *In LNCS:*1953, pages 392-405. Springer, 2000.