PARALLEL DISTRIBUTED-MEMORY PARTICLE METHODS FOR ACQUISITION-RATE SEGMENTATION AND UNCERTAINTY QUANTIFICATIONS OF LARGE FLUORESCENCE MICROSCOPY IMAGES

A dissertation submitted to
TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK

for the degree of
Doktor-Ingenieur (Dr.-Ing.)

presented by
YASER AFSHAR
M.Sc. Mechanical Engineering
born on April 26th, 1980
citizen of Tehran, Iran

accepted on the recommendation of
PROF. IVO F. SBALZARINI
PROF. MICHAEL SCHRÖDER
PROF. GENE MYERS
PROF. LARS HUFNAGEL

Date of defense: October 17th, 2016

Declaration of Authorship

This thesis is a presentation of my own original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature. The work was done at the Technische Universität Dresden, and project carried out at the Center for Systems Biology Dresden and the Max Planck Institute of Molecular Cell Biology and Genetics under the supervision of Prof. Dr. Ivo F. Sbalzarini. I hereby declare that this thesis has not been submitted before to any institution for assessment purposes.

Dresden, July 11, 2016 Yaser Afshar

Acknowledgments

This thesis would not have been possible without the support of many people at MPI-CBG and beyond. First and most of all, I would like to express my special appreciation to Ivo, for giving me the opportunity to do my Ph.D. in the MOSAIC group. Ivo, you have been always amazingly open, positive, and supportive. I would like to thank you for sharing your enthusiasm and your ideas for our work which has been very contagious and motivating. I would like to thank you for encouraging my research and for helping me to establish my career as a research scientist. Your advice on both research as well as on my career has been priceless. Thank you!

I would also like to thank my thesis committee members, Prof. Dr. Gene Myers and Prof. Dr. Michael Schröder, who have been my thesis advisory members after I started my Ph.D. thesis at MPI-CBG. I am thankful to all for the insightful comments and sincere encouragement during the last four years. Furthermore, I am grateful to my external reviewer, Dr. Lars Hufnagel, for accepting to review this thesis.

I want to thank sincerely the members of the MOSAIC group, who have contributed immensely to my personal and professional time at MPI-CBG. The group has been a source of friendships as well as good advice and collaboration. I want to thank all the present and past members of MOSAIC, with whom, I had the pleasure to work and to interact. They all

contributed to the stimulating and friendly atmosphere in many ways and made my Ph.D. adventure a very pleasant experience indeed.

Big cheers go to my awesome friends at MOSAIC. Thanks to Pietro and Bevan for the countless challenging and fruitful discussions. Millions of thanks go to Josefine for the good moments we had working together and most importantly for being a wonderful friend in every way. A special thank you to Ulrik, for sharing his expertise on Apple hardware and for introducing me to rock climbing and vertical adventures. I am also glad to have worked with Rajesh, I always enjoyed our discussions and benefited from his scientific knowledge. I enjoyed all the discussions with Rajesh and Pietro in the lab. It was a pleasant experience to work with Alex. I enjoyed Alex's scientific mind and the way he shares it. I was truly pleased to share the office with Krzysztof. I appreciate his honesty and his sense of humor. I enjoyed the discussions with Ömer on parallel programming and parallelization technology. It was very pleasant to work with Sophie. I acknowledge her honesty and calmness and all the good moments we shared together at MOSAIC. I'd like to thank Alejandro for contributing to the good mood in the group. I am also grateful to Anastasia, Benjamin, and Michael for the great moments we had working together and for all our discussions. I am also thankful to Gong, for many valuable discussions that helped me understand image processing when I joined the group and when this research area was still new to me.

I am indebted to Ms. Carolyn Fritzsche from the international office of MPI-CBG for her support in my personal and professional life, and to Ms. Antje Walter who helped me and my wife in our marriage through each and every step with the paper work and the German bureaucracy. I am also thankful to our kind and helpful secretary, Mrs. Susann Gierth.

Moreover, I would like to acknowledge the financial, academic and technical support of the Max Planck Institute of Molecular Cell Biology and Genetics (MPI-CBG) and its staff.

Above all, special thanks go to my family. I have an amazing family, unique in many ways. Words cannot express how grateful I am for all the sacrifices that they made for me. Without them, it would not have been possible for me to study and carry out this thesis. Thank you for encouraging me in all of my pursuits and for inspiring me to follow my dreams. I am especially

grateful to my parents, who supported me in every way. I always knew that they believed in me and wanted the best for me. They have cherished with me every great moment and supported me whenever I needed it. I'm profoundly grateful.

All my love goes to my wife. Fafa, thank you for starting this journey with me and always backing me in all difficulties, even though we were half a globe apart. You encouraged me to do the best and helped me to follow my dreams. Thanks for your love and support. You are my eternal sunshine.

Yaser

Abstract

Modern fluorescence microscopy modalities, such as light-sheet microscopy, are capable of acquiring large three-dimensional images at high data rate. This creates a bottleneck in computational processing and analysis of the acquired images, as the rate of acquisition outpaces the speed of processing. Moreover, images can be so large that they do not fit the main memory of a single computer. Another issue is the information loss during image acquisition due to limitations of the optical imaging systems . Analysis of the acquired images may, therefore, find multiple solutions (or no solution) due to imaging noise, blurring, and other uncertainties introduced during image acquisition.

In this thesis, we address the computational processing time and memory issues by developing a distributed parallel algorithm for segmentation of large fluorescence-microscopy images. The method is based on the versatile Discrete Region Competition (Cardinale et al., 2012) algorithm, which has previously proven useful in microscopy image segmentation. The present distributed implementation decomposes the input image into smaller sub-images that are distributed across multiple computers. Using network communication, the computers orchestrate the collective solving of the global segmentation problem. This not only enables segmentation of large images (we test images of up to 10^{10} pixels), but also accelerates segmentation to match the time scale of image acquisition. Such acquisition-rate image segmentation is a prerequisite for the smart microscopes of the future and enables online data inspection and interactive experiments.

Second, we estimate the segmentation uncertainty on large images that do not fit the main memory of a single computer. We therefore develop a distributed parallel algorithm for efficient Markov-chain Monte Carlo Discrete Region Sampling (Cardinale, 2013). The parallel algorithm provides a measure of segmentation uncertainty in a statistically unbiased way. It approximates the posterior probability densities over the high-dimensional space of segmentations around the previously found segmentation.

Zusammenfassung

Moderne Fluoreszenzmikroskopie, wie zum Beispiel Lichtblattmikroskopie, erlauben die Aufnahme hochaufgelöster, 3-dimensionaler Bilder. Dies führt zu einen Engpass bei der Bearbeitung und Analyse der aufgenommenen Bilder, da die Aufnahmerate die Datenverarbeitungsrate übersteigt. Zusätzlich können diese Bilder so groß sein, dass sie die Speicherkapazität eines einzelnen Computers überschreiten. Hinzu kommt der aus Limitierungen des optischen Abbildungssystems resultierende Informationsverlust während der Bildaufnahme. Bildrauschen, Unschärfe und andere Messunsicherheiten können dazu führen, dass Analysealgorithmen möglicherweise mehrere oder keine Lösung für Bildverarbeitungsaufgaben finden.

Im Rahmen der vorliegenden Arbeit entwickeln wir einen verteilten, parallelen Algorithmus für die Segmentierung von speicherintensiven Fluoreszenzmikroskopie-Bildern. Diese Methode basiert auf dem vielseitigen “Discrete Region Competition” Algorithmus (Cardinale et al., 2012), der sich bereits in anderen Anwendungen als nützlich für die Segmentierung von Mikroskopie-Bildern erwiesen hat. Das hier präsentierte Verfahren unterteilt das Eingangsbild in kleinere Unterbilder, welche auf die Speicher mehrerer Computer verteilt werden. Die Koordinierung des globalen Segmentierungsproblems wird durch die Benutzung von Netzwerkkommunikation erreicht. Dies erlaubt die Segmentierung von sehr großen Bildern, wobei wir die Anwendung des Algorithmus auf Bildern mit bis zu 10^{10} Pixeln demonstrieren. Zusätzlich wird die Segmentierungsgeschwindigkeit erhöht und damit vergleichbar mit der Aufnahmerate des Mikroskops. Dies ist eine Grundvoraussetzung für die intelligenten Mikroskope der Zukunft, und es erlaubt die Online-Betrachtung der aufgenommenen Daten, sowie interaktive Experimente.

Wir bestimmen die Unsicherheit des Segmentierungsalgorithmus bei der Anwendung auf Bilder, deren Größe den Speicher eines einzelnen Computers übersteigen. Dazu entwickeln wir einen verteilten, parallelen Algorithmus für effizientes Markov-chain Monte Carlo “Discrete Region Sampling” (Cardinale, 2013). Dieser Algorithmus quantifiziert die Segmentierungsunsicherheit statistisch erwartungstreu. Dazu wird die A-posteriori-Wahrscheinlichkeitsdichte über den hochdimensionalen Raum der Segmentierungen in der Umgebung der zuvor gefundenen Segmentierung approximiert.

CONTENTS

DECLARATION OF AUTHORSHIP i

ACKNOWLEDGEMENTS iii

ABSTRACT I

ZUSAMMENFASSUNG III

1 INTRODUCTION 1

THESIS OUTLINE AND CONTRIBUTIONS 4

2 PRELIMINARIES 9

2.1 THE BAYESIAN IMAGE SEGMENTATION PARADIGM 9

2.2 REVIEW OF DISCRETE REGION COMPETITION 11

2.3 INTRODUCTION TO MCMC 16

2.3.1 The Metropolis-Hastings algorithm 21

2.4 Review of Discrete Region Sampling 22

2.5 Distributed computing 27

2.5.1 Middleware for distributed computing: PPM 28

3 Parallel distributed-memory Discrete Region Competition algorithm 29

3.1 Introduction 29

3.2 Data distribution by domain decomposition 29

3.2.1 Initialization 30
3.2.2 Parallel connected-component labeling 33
3.2.3 Parallel contour propagation 41
3.2.4 Parallel topology processing and data-structure update 50

3.3 Results 55

3.3.1 Correctness of the distributed algorithm 56
3.3.2 Efficiency of the distributed algorithm 66
3.3.3 Application to acquisition-rate segmentation of 3D light-sheet microscopy data 71

3.4 Summary 80

4 Parallel distributed-memory Discrete Region Sampling algorithm 83

4.1 Introduction 83

4.1.1 Previous approaches 84

4.2 Algorithm 89

4.2.1 Domain decomposition 89
4.2.2 Checkerboard decomposition 90
4.2.3 Parallel Pseudo random number generation 92
4.2.4 Iteration structure 94
4.2.5 Data structure 101

4.3 RESULTS 102

4.4 SUMMARY 103

5 CONCLUSIONS AND FUTURE WORK 105

CURRICULUM VITAE 117

NOMENCLATURE

List of Acronyms

2D	two-dimensional
3D	three-dimensional
AR	acceptance rate
BG	background
CMOS	complementary metal-oxide-semiconductor
CUDA	a parallel computing platform and programming model
DRC	discrete region competition
DRS	discrete region sampling
DSL	domain-specific programming language
eDSL	embedded domain-specific programming language
FBR	forward-backward ratio
FFTW	fastest Fourier transform in the west
FG	foreground
GB	gigabyte
GFP	green fluorescent protein
GPU	graphics processing unit
HKA	Hoshen-Kopelman algorithm
HR	Hastings ratio
ITK	insight segmentation and registration toolkit

MAP	maximum-a-posteriori
MC	Monte Carlo
MCMC	Markov-chain Monte Carlo
MH	Metropolis-Hastings
MPI-CBG	Max Planck Institute of Molecular Cell Biology and Genetics
MPI	message passing interface
MTRNG	Mersenne Twister random number generator
PC	piecewise constant
PPM	parallel particle mesh library
PPML	parallel particle mesh language
PS	piecewise smooth
RMA	remote memory access
RNG	pseudo random number generator
SPIM	selective plane illumination microscopy
SRNG	Saru random number generator

Greek Characters

α_{ij}	transition probability
Δr_{max}	edge length of the cell
$\Delta\mathcal{E}$	energy difference
Γ	segmentation contour
Γ'	perturbed segmentation contour
Γ_i	contour of region i
λ	regularization parameter
Ω	image domain
$\pi(.)$	target distribution
σ	standard deviation of a Gaussian
τ	hitting time

Latin Characters

bc	boundary cell
C	set of states
$\mathcal{C}_c$	checkerboard set
$\mathcal{C}_b$	array of border cell indices
$\mathcal{C}_i$	array of interior cell indices
$\mathcal{C}_n$	displacement array of interior cells
d	image dimension
d_i	period of a state i
$\mathcal{E}$	energy function
E	expectation value
$\bar{f}$	an estimate of the expectation of f
f_{ij}^n	probability of chain starting at i entering j for the first time at the nth step
G_P	undirected graph on processor P
G_P^i	interior sub-graphs on processor P
G_P^b	boundary sub-graphs on processor P
I	image
L^1	L1-norm or least absolute deviation
L^2	L2-norm or least squares deviation
L	label image
l'	candidate label of a particle
L_0	initial label image
L	length of sub-image border
$\mathcal{L}$	collection of segmentations
M	number of regions
$\mathcal{M}$	hash map
$\mathcal{P}_f$	set of floating particles
p_{ij}	transition probability to move from state i to state j
$\mathcal{P}$	set of regular particles
P	number of processors
P	transition matrix
$\mathbf{p}$	a particle

q_f	off-boundary sampling probability
$\mathbf{q}$	a particle
$\mathbb{R}$	the set of real numbers
S	finite state space
X_i	image region, set of discrete points
X_n	random variable at time or position n
X'	proposal state

Special symbols

∇	Nabla operator
$\mathcal{O}$	complexity function, “big-O notation”
$\rightarrow$	a state is accessible from the other
$\leftrightarrow$	states communicate with each other
$\subset$	sub-set
$\in$	set membership

List of Figures

2.1 Illustration of two regions 12

2.2 Illustration of causally dependent moves 14

2.3 Illustration of dependency graph construction 15

2.4 Illustration of a region 24

2.5 Illustration of a topological trap 25

3.1 Domain decomposition and distribution of an image 30

3.2 Ghost layers 31

3.3 Region label initialization 32

3.4 Distributed parallel connected-component labeling 34

3.5 Processors independently label the local part 35

3.6 The border relabeling 36

3.7 Sending border to the master node 38

3.8 An array of unique pairs 39

3.9 The global array of all unique pairs 40
3.10 The master-slave approach 43
3.11 Distributed sub-graph algorithm 44
3.12 Energy evolution 47
3.13 Comparison of the results 48
3.14 Local pixel oscillations 50
3.15 Distributed region split and merge algorithm 51
3.16 Activated ("hot") boundary particles 52
3.17 Distributed segmentation of a synthetic image 57
3.18 Energy evolution 58
3.19 Distributed segmentation of a natural-scene image 59
3.20 Distributed segmentation of fluorescently labeled cell nuclei 60
3.21 Parallel segmentation of a synthetic image 62
3.22 Energy evolution 63
3.23 Distributed segmentation of a natural-scene image 64
3.24 Distributed segmentation of zebrafish primordial germ cells 65
3.25 Maximum-intensity projections 67
3.26 Weak scaling parallel efficiency 69
3.27 Strong scaling speedup 70
3.28 Application to acquisition-rate segmentation 73
3.29 Comparison of PC and PS image models 76
3.30 Application to segmenting zebrafish vasculature 79

4.1 An image with four sub-images 86

4.2 Sampling two particles simultaneously in two sub-images . . 87

4.3 Image distributed to sixteen sub-images 90

4.4 Illustration of boundary and interior cells 95

4.5 Illustration of the five sub-sweeps over cells 97

4.6 Communications after sampling 100

CHAPTER ONE

INTRODUCTION

Modern fluorescence microscopes with high-resolution cameras are capable of acquiring large images at a fast rate. Data rates of 1 GB/s are common with CMOS cameras, and the three-dimensional (3D) image volumes acquired by light-sheet microscopy (Huisken et al., 2004) routinely exceed tens of gigabytes per image, and tens of terabytes per time-lapse experiment (Huisken and Stainier, 2007; Preibisch et al., 2010; Schmid et al., 2013). This defines new challenges in handling, storing, and analyzing the image data, as image acquisition outpaces analysis capabilities.

Ideally, the images are analyzed during acquisition with analysis times that are smaller than the time until the next image is acquired. This "real-time" image analysis not only alleviates the data bottleneck, but is also a prerequisite for smart microscopes that optimize the acquisition of the next image based on the contents of the current image (Scherf and Huisken, 2015). Real-time segmentation also enables interactive experiments where, e.g., optical manipulation and tracking become feasible in a developing embryo (Amat et al., 2014).

Real-time, or more precisely acquisition-rate, segmentation of large images is usually hindered by the memory requirements of the image data and the analysis algorithm. Segmenting an image requires about 5 to 10 times more memory than the raw image data (Caselles et al., 1997; Beare and Lehmann, 2006; Al-Kofahi et al., 2010). This means that in order to segment a 30 GB 3D light-sheet microscopy image, one would need a computer with 150 to 300 GB of main memory. Image segmentation at acquisition rate has hence mainly been achieved for smaller images (Stegmaier et al., 2014). For example, segmenting a $2048 \times 2048 \times 400$ pixel image of stained nuclei, which translates to about 3 GB file size at 16 bit depth, required more than 32 GB of main memory (Stegmaier et al., 2014).

Acquisition-rate processing of large images has so far been limited to low-level image processing, such as filtering or blob detection. Pixel-by-pixel low-level processing has been accelerated by Olmedo et al. (2012) using CUDA as a parallel programming tool on graphics processing units (GPUs). In their work, pixel-wise operations are applied to many pixels simultaneously, rather than sequentially looping through pixels. While such GPU acceleration achieves high processing speeds and data rates, it is limited by the size of the GPU memory, which is in general smaller than the main memory. Another approach is to distribute different images to different computers. In a time-lapse sequence, every image can be sent to a different computer for processing. Using 100 computers, every computer has 100 frames time to finish processing its image, until it receives the next one. While this does not strictly fulfill the definition of acquisition-rate processing (e.g., it would not be useful for a smart microscope), it improves data throughput by pipelining. Galizia et al. (2015) have demonstrated this in the parallel image processing library *GEnoa*, which runs on computer clusters using the Message Passing Interface (MPI) to distribute work, but it also runs on GPUs and GPU clusters. This library focuses on low-level image processing. Both GPU acceleration and embarrassingly parallel work-farming approaches are, however, unable to provide acquisition-rate high-level image analysis of single large images or time series comprised of large images.

High-level image analysis in fluorescence microscopy is mostly concerned with image segmentation (Aubert and Kornprobst, 2006; Cremers et al., 2007). In image segmentation, the task is to detect and delineate objects represented in the image. This is a high-level task, which cannot be done in a pixel-independent way. It also cannot be formulated as a shader or filter, rendering it hard to exploit the speed of GPUs. Finally, as outlined above, high-level image analysis of large images quickly exceeds the main memory of a single computer. This memory limitation can be overcome by sub-sampling the image, for example coarse-graining groups of pixels to *super-pixels*. This has been successfully used for acquisition-rate detection of nuclei and lineage tracking from large 3D images (Amat et al., 2014). The generation of super-pixels only requires low-level operations, where the high-level analysis is done on the reduced data. While this effectively enables acquisition-rate high-level analysis, it does not provide single-pixel resolution and is somewhat limited to the specific application of lineage tracing.

Pixel-accurate high-level analysis of large images can be achieved by splitting each image into smaller sub-images and distributing them across multiple computers or memories, thus distributing the data and the work. The computers then work in parallel, each on its sub-image. They communicate over a network interconnect in order to collectively solve the same high-level image-analysis problem that a single computer would have solved. However, since the data are distributed, the solution is available faster, and arbitrarily large images can be accommodated by distributing across more computers. This is the hallmark of *distributed-memory parallelism*. This strategy enables segmentation of large images, and accelerates segmentation to match the time scale of image acquisition. However, image segmentation constitutes an ill-posed problem. Thus, the segmentation might be uncertain without providing any information about the uncertainty in the solution. Providing confidence intervals and uncertainties of the analysis results would provide more information and enable higher-level reasoning about the solution. Therefore, it is critical to capture the uncertainty of the segmentation before declaring the final decision or storing the results, in order to decide whether an observed difference between samples is real, or a processing artifact. In Bayesian image analysis, the distribution of potential results and a representative collection of them can be estimated by drawing samples from the posterior distribution. Representative samples from the posterior provide additional insight into the robustness of the segmentation. However, the sampling approach, no matter how smart, is computationally expensive and inherently sequential. We explore the distributed-memory approaches that relax these issues for data that do not fit the memory of a single computer.

Thesis Outline and Contributions

In this thesis, we address both processing-time and memory issues by developing a distributed parallel framework for segmentation and uncertainty quantification of large fluorescence microscopy images. The method is based on the versatile Discrete Region Competition (Cardinale et al., 2012) algorithm, which has previously proven useful in microscopy image segmentation. The present distributed implementation decomposes the input image into smaller sub-images that are distributed across multiple computers. Using network communication, the computers orchestrate the collective solving of the global segmentation problem. This not only enables segmentation of large images (we test images of up to 10^{10} pixels), but also accelerates segmentation to match the time scale of image acquisition. Such acquisition-rate image segmentation is a prerequisite for the smart microscopes of the future and enables online data inspection and interactive experiments.

We also address both the processing-time and memory issues of Markov-chain Monte Carlo algorithms for assessing the segmentation quality and robustness. The method is based on a novel, efficient, particle-based Metropolis-Hastings algorithm (Cardinale, 2013), called discrete region sampling (DRS). Again, the present distributed implementation decomposes the input image into smaller sub-images that are distributed across multiple computers. Using network communication, the computers orchestrate the collective sampling from the posterior distribution of the defined segmentations on the observed image in a statistically unbiased manner.

Chapter 2: Preliminaries

We begin by discussing the relevant background material in Chapter. 2. After motivating the Bayesian formulation of image analysis, we introduce the DRC image-segmentation algorithm. We give a brief introduction to

Markov chain theory, followed by a review of previous shape sampling approaches. We provide an introduction to distributed computing and conclude with an overview of the parallel particle mesh library (PPM) as a middleware for distributed particle methods.

Chapter 3: Parallel distributed-memory Discrete Region Competition algorithm

We present a distributed-memory parallel design and implementation of the generic image-segmentation algorithm DRC. The present implementation scales to large images. Here, we test images of size up to $8192 \times 8192 \times 256 = 1.7 \cdot 10^{10}$ pixels, corresponding to 32 GB of data per image at 16 bit depth. We show that distributing an image across 128 processors enables acquisition-rate segmentation of large light-sheet microscopy images of *Drosophila* embryos. *Discrete Region Competition* (DRC) (Cardinale et al., 2012) is a general-purpose model-based segmentation method. It is not limited to nucleus detection or any other task, but solves generic image-segmentation problems with pixel accuracy. The method is based on using computational particles to represent image regions. This particle-method character renders the computational cost of the method independent of the image size, since it only depends on the total contour length of the segmentation. Storing the information on particles effectively reduces the problem from 3D to 2D (or from 2D to 1D). Moreover, the particle nature of the method lends itself to distributed parallelism, as particles can be processed concurrently, even if pixels cannot. In terms of computational speed, DRC has been shown competitive with fast discrete methods from computer vision, such as multi-label graph-cuts (Cardinale et al., 2012; Delong et al., 2012). Single-processor DRC has previously been demonstrated on 2D and 3D images using a variety of different image models, including piecewise constant, piecewise smooth, and deconvolving models (Cardinale et al., 2012).

The piecewise constant and piecewise smooth models are also available in the present distributed-memory parallel implementation. This makes available a state-of-the-art generic image segmentation toolbox for acquisition-rate analysis of large images that do not need to fit the memory of a single computer. The main challenge in parallelizing the DRC algorithm is to ensure global topological constraints on the image regions. These are required in order for regions to remain closed or connected. The main algorithmic contribution of the present work is hence to propose a novel distributed algorithm for the independent-sub-graph problem. Moreover, we present a new parallel algorithm for connected-component labeling in 2D and 3D images. The presented algorithm is both memory and computationally efficient and scales to large numbers of processors.

The algorithmic solutions presented in Chapter 3 ensure that the final result computed is the same that would have been computed on a single computer, and that the network communication overhead is kept to a minimum, hence ensuring scalability to large images.

Since each computer only stores its local sub-image, information needs to be communicated between neighboring sub-images in order to ensure global consistency of the solution. Since DRC is a particle method, we use the Parallel Particle Mesh (PPM) library (Sbalzarini et al., 2006; Awile et al., 2010, 2013) for work distribution and orchestration of the parallel communication.

We then present the main algorithmic contribution that made this possible: the distributed independent-sub-graph algorithm. We demonstrate correctness of the parallel implementation by comparing with the sequential reference implementation of DRC (Cardinale et al., 2012), as available in ITK (Ibanez et al., 2005). We then benchmark the scalability and parallel efficiency of the new parallel implementation on synthetic images, where the correct solution is known. Finally, we showcase the use of the present implementation for acquisition-rate segmentation of light-sheet fluorescence microscopy images.

CHAPTER 4: PARALLEL DISTRIBUTED-MEMORY DISCRETE REGION SAMPLING ALGORITHM

Accurately and robustly segmenting an image is a challenging task. Usually, the segmentation is not unique and can be locally uncertain. Even using a global approach and providing bounds on the final energy of a solution does not provide information about its quality. This means that many cases require user interaction, for example by iterating over the results and correcting mistakes. This process is usually too time-consuming or does not yield reproducible results, especially at places in the image with low signal-to-noise ratio.

Estimating the uncertainty and robustness of a segmentation can reduce and guide user interaction. It also enables statistical tests conveying the information about the segmentation reliability. Cardinale (2013) presented a method for sampling from the posterior distribution of explicitly or implicitly defined segmentations, conditioned on the observed image. In this approach, a discrete deformable model evolves, such that the sampled segmentations approximate the posterior distribution of possible segmentations. This allows assessing segmentation robustness. The presented particle-based Metropolis-Hastings algorithm, called *Discrete Region Sampling*, has been compared with a state-of-the-art algorithm by Chang and Fisher (2012) in terms of solution quality and has been shown competitive w.r.t. computation time.

In Chapter 4, we present a distributed-memory parallel extension of this sampling approach. Because of the inherently sequential nature of the sampling approach, parallelization is challenging. The primary challenge in parallelizing the DRS algorithm is to update a large number of particles in a statistically unbiased way and to guarantee the detailed balance, which is a sufficient condition for convergence. The main algorithmic contribution of the present work is to propose and implement a novel parallel distributed algorithm for efficiently sampling from the posterior distribution of defined segmentations on the observed image in a statistically correct manner.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

Finally, in Chapter 5, we summarize the present work and the contributions of this thesis. We provide recommendations for extensions and future work pertaining to the developed parallel algorithms for segmentation and uncertainty quantification of fluorescence-microscopy images.

CHAPTER

TWO

PRELIMINARIES

We review relevant background materials for this thesis. We begin by motivating the Bayesian formulation for image analysis; then we introduce the image segmentation algorithm. We give a brief introduction to Markov chain theory, followed by a review of an efficient shape sampling approach. We provide an introduction to distributed computing and conclude with an overview of the parallel particle mesh library (PPM) as a middleware for distributed computing.

2.1 THE BAYESIAN IMAGE SEGMENTATION PARADIGM

Image segmentation has been a fundamental topic in computer vision and image understanding and Bayesian inference offers an elegant mathematical formulation of it (Winkler, 2003). Let us denote the observed image by I and Γ the segmentation of the image. The goal of segmentation (here we are primarily concerned with the segmentation of objects in fluorescence microscope images) is to reconstruct the objects shown in the image from a given observed image. In this sense, segmentation is an inverse problem. Segmentation is also “ill-posed,” which means that there may not be a unique solution. Several solutions (or no solution) can be due to imaging noise, blurring, and other uncertainties introduced during image acquisition.

In image segmentation, the goal is to assign a label to each pixel at pixel grid (i, j) in I, indicating to which object or region that pixel belongs. The ill-posedness can be overcome by introducing application-specific prior information about the solution in order to constrain the problem. The prior knowledge can, for example, be a smoothness constraint on the segmentation. Hence, it is required to estimate the segmentation given the observed image data. This estimation can be done by maximizing the probability distribution,

$$P(\Gamma|I), \tag{2.1}$$

where probability represents uncertainty about the values of parameters. In the Bayesian framework, this probability is represented as :

$$P(\Gamma|I) = \frac{P(I|\Gamma) \cdot P(\Gamma)}{P(I)}, \tag{2.2}$$

where $P(I|\Gamma)$ is the likelihood, $P(\Gamma)$ is the prior over the segmentation and $P(\Gamma|I)$ is the posterior distribution. The likelihood is the conditional probabilistic model linking the unknown objects to the observation I. It expresses how likely it is to observe the image I under a given segmentation Γ.

The prior distribution is a marginal probabilistic model that introduces some physical understanding or intuition into the process. It measures how likely a certain segmentation Γ is. Some priors may impose a smoothness or regularization constraint, for example, the one proposed by Mumford and Shah (1989), which penalizes the Euclidean contour length $|\Gamma|$. Other priors may impose shape characteristics and penalize deviations of the segmentation from the template shape (Kim and Kim, 2000; Belongie et al., 2002; Osada et al., 2002; Zhang and Lu, 2004; Rose et al., 2009).

The posterior distribution expresses the state of knowledge about the model after estimating the parameters. One aims to estimate the segmentation parameters that maximize this probability. This is known as the Maximum-A-Posterior; (MAP) estimation. MAP estimation involves both the likelihood and the prior terms :

$$\max_{\Gamma} \ P(\Gamma|I). \tag{2.3}$$

MAP estimation aims to find the segmentation Γ that maximizes the posterior probability based on the observed image I. We can reformulate the MAP estimation as a minimization problem. Taking the negative of the natural logarithm of both sides of 2.2, we get

$$-\,logP(\Gamma|I) = -logP(I|\Gamma) - logP(\Gamma) + C, \tag{2.4}$$

which is the negative posterior *log likelihood*, and this negative log likelihood is considered as energy. So, maximizing the posterior probability can be done by simultaneously minimizing two energy terms (the value of constant C does not matter during energy minimization). They correspond to the likelihood and prior and called *external* and *internal* energy, respectively. In a discrete setting, and for a traditional gradient-based energy minimization approach, the energy gradient becomes an energy difference $\Delta\mathcal{E}$ when deforming Γ to Γ'. The basic idea is to use a method which allows a segmentation Γ to deform to Γ' so as to minimize a given energy functional in order to produce the desired segmentation.

Here, we review a discrete multi-region gradient descent algorithm to optimize the posterior (minimize energy) locally over possible segmentations. This general-purpose image-segmentation method is called Discrete Region Competition (Cardinale et al., 2012).

2.2 Review of Discrete Region Competition

Since the introduction of active contours (Kass et al., 1988), deformable models have extensively been used for image segmentation. They are characterized by a geometry representation and an evolution law (Montagnat et al., 2001). Thorough reviews of deformable models can be found in Montagnat et al. (2001); Zhang and Lu (2004). The geometry representation of the evolving contours in the image can be continuous or discrete, and in either case implicit (also called "geometric models") or explicit (also called "parametric models") (Xu et al., 2000).

Inspired by discrete level-set methods (Shi and Karl, 2008), and motivated by the wish to delineate different objects in an image as individual regions, Cardinale et al. (2012) presented a discrete deformable model where the contour is represented by computational particles placed on the pixel grid. This is illustrated in Fig. 2.1 and provides a geometry representation that is both explicit and implicit (Shi and Karl, 2008). During the iterative segmentation process, the particles migrate to neighboring pixels and hence deform the contour. This migration is driven by an energy-minimization flow. Additional topological constraints ensure that contours remain closed and/or connected. The algorithm is a discrete version of Region Competition (Zhu and Yuille, 1996), which converges to a locally optimal solution. It is called *Discrete Region Competition* (DRC), since particles from adjacent regions compete for ownership over pixels along common boundaries.

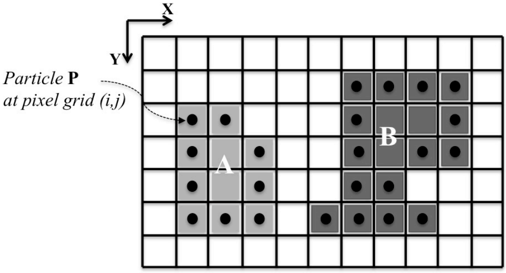


Figure 2.1: Illustration of 2 regions (A, light gray and B, dark gray) in a 2D digital image (grid). Pixels in the background region are white. Particles are shown as black filled circles. They represent the regions by marking their outlines. Shaded pixels without a particle are interior points of the respective region.

The algorithm partitions a digital image domain $\Omega \subset \mathbb{R}^d$ (the dimension $d = 2$ or 3) into a background (BG) region X_0 and $(M-1) > 0$ disjoint foreground (FG) regions $X_i, i = 1, \cdots, M-1$, bounded by contours Γ_i, $i =$

$1, \cdots, M-1$ (Cardinale et al., 2012).

FG regions are constrained to be connected sets of pixels. The void space around the FG regions is represented by a single BG region, which needs not be connected. Connectivity in the FG regions is defined by a face-connected neighborhood, i.e., 4-connected in 2D and 6-connected in 3D. The BG region then has to be 8-connected in 2D and 18 or 26-connected in 3D (Ségonne, 2005). Imposing the topological constraint that FG regions have to be connected sets of pixels regularizes the problem to the extent where the number of regions can be jointly estimated with their intensities and contours (Cardinale et al., 2012).

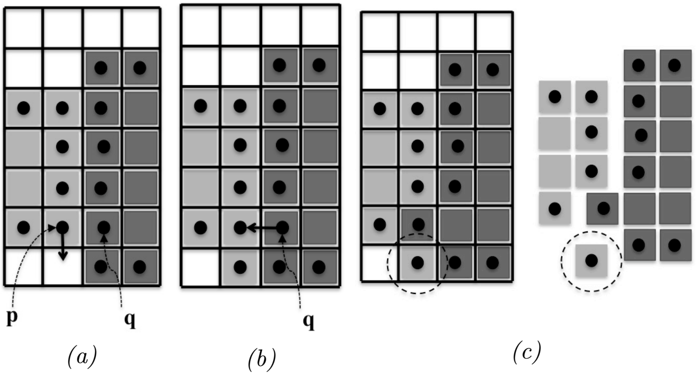
The evolving contour is represented by computational particles as shown in Fig. 2.1. The algorithm advances multiple particles simultaneously in a processing order that does not depend on particle indexing. This ensures convergence to a result that is independent of the order in which particles are visited. Connectedness of the evolving contours is ensured by topological control. The motion of the particles is driven by a discrete energy-minimization flow that locally minimizes the segmentation energy functional (Cardinale et al., 2012):

$$\mathcal{E} = \mathcal{E}_{\text{data}} + \lambda \mathcal{E}_{\text{length}} + \alpha \mathcal{E}_{\text{merge}} \,. \tag{2.5}$$

Here, λ and α are regularization parameters trading off the weights of the contour length and region-merging priors, respectively. The first term is the negative logarithm of the likelihood and measures how well the current segmentation fits (or explains) the image. The specific forms of the three terms depend on the image model, imaging model, and object model used (Cardinale et al., 2012).

The above energy is minimized by approximate gradient descent. The gradient is approximated by the energy difference incurred by a particle move. Particles are then moved in order of descending energy reduction using a rank-based optimizer, hence ensuring that the result is independent of particle ordering (Cardinale et al., 2012). Since regions may dynamically fuse and split during energy minimization, the algorithm is able to detect and segment a previously unknown and arbitrary number of regions.

The algorithm starts from a hypothetical segmentation as an initialization (frequently: local intensity maxima or an initial thresholding) and then refines the segmentation in iterations until no further improvement can be achieved by any particle move. In each iteration, every particle finds the set of adjacent pixels it could potentially move to. It then computes the energy differences of all possible moves. Moves that lead to topological violations are pruned from the list. Then, a graph of causally dependent moves is constructed. An example of causal dependency is illustrated in Fig. 2.2, where the possible moves of particle **p** depend on the move of particle **q**. Assume that the energetically most favorable move for particle **p** is downward (Fig. 2.2a). If the energetically most favorable move of particle **q** is to go left (Fig. 2.2b), this violates the topological constraint that the light-gray region has to be a 4-connected set of pixels (Fig. 2.2c). Simply executing the energetically most favorable move for each particle could hence lead to topological violations. In the situation shown in Fig. 2.2, only one of the two particles can execute its most favorable move, constraining the possible moves of the other. In the present greedy descent scheme, the move that leads to the largest energy decrease has priority.

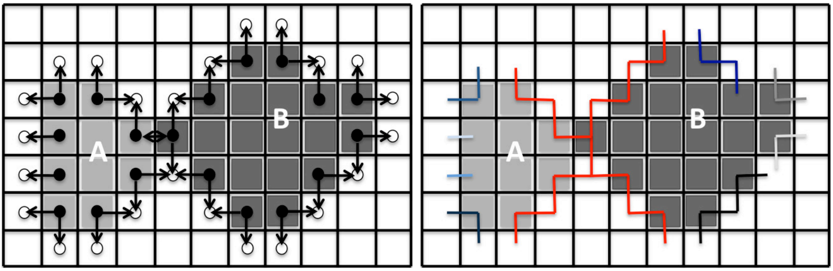


(a) (b) (c)

Figure 2.2: Illustration of causally dependent moves. Assume that the energetically most favorable moves are for particle **q** *to move down (a) and particle* **q** *left (b). If both moves are executed, the light gray region is not connected any more, hence violating the topological constraint (c). The moves of the two particles hence causally depend on each other.*

In order to find the set of moves that can be executed concurrently, we build a graph of all such causal dependencies and sort them by energy. Figure 2.3 illustrates the construction of this undirected graph of causal dependencies.

It starts from enumerating all possible moves for all particles (Fig. 2.3a). Shrinking a region is done by removing the respective particle and inserting new boundary particles. This is irrelevant for the dependency graph. The directionality of the moves is also irrelevant and is removed, yielding a set of undirected edges. A vertex is introduced wherever two edges meet in any pixel. This defines the final graph (Fig. 2.3b). Moves that are connected by a path in the graph are causally dependent. Maximal connected sub-graphs of the final graph (highlighted by different colors in Fig. 2.3b) hence correspond to dependent sets of moves. They can extend across several particles, defining long-range chains of dependence.



(a) *(b)*

Figure 2.3: Illustration of dependency graph construction. (a) All possible moves are enumerated for all particles. (b) The undirected graph of causal dependences is obtained by removing directionality and joining edges that share a common pixel. The maximal connected sub-graphs are represented by different colors.

Each maximal connected sub-graph can be processed independently. While the moves within a maximal connected sub-graph are causally dependent, there are no dependencies across different maximal connected sub-graphs. In order to find the energetically most favorable set of moves that can be executed simultaneously, the edges in each maximal connected sub-graph are sorted by energy difference. In each sub-graph, the move that leads to the largest decrease in energy is executed.

Splits and fusions of FG regions are topological changes that are allowed by the energy. They are detected using concepts from digital topology (Bertrand, 1994; Ségonne, 2005; Lamy, 2007; Shi and Karl, 2008; Cardinale et al., 2012) and accepted if energetically favorable. The BG region is allowed to arbitrarily change its topology.

2.3 Introduction to MCMC

To provide a measure of segmentation uncertainty or solution robustness, we introduce an efficient Markov-chain Monte Carlo method to approximate the posterior probability densities over the high-dimensional space of segmentations around the solution found by DRC. We start with a brief overview of Markov chain theory.

Markov-chain Monte Carlo (MCMC) algorithms were first developed for applications in statistical physics and later in the statistics community. The *Metropolis algorithm* has been introduced in the classic paper of Metropolis et al. (1953). Later on, Geman and Geman (1984) presented a special case of the algorithm and applied it to image processing. Their influential paper introduced the method for Bayesian applications and started new works on the *Gibbs sampler*.

In this section, we briefly review the key concepts behind Markov-chain Monte Carlo methods in order to make this thesis self-contained. All definitions and theorems in this subsection are selected and adapted from Ó Ruanaidh and Fitzgerald (1996) and Bremaud (1999), where the full descriptions and proofs can be found.

Suppose, we have a probability distribution $\pi(.)$. For example consider $\pi(.)$ is the posterior $P(\Gamma|I)$ of segmentations given the image I (see equation 2.2). We do not know this distribution explicitly. Also, it is a complicated distribution and non-trivial to sample from. Sampling can help avoid local minima. Also, sampling contributes to exploring the configuration space for a full description of the posterior, which results in information about estimation uncertainty.

Markov Chain Monte Carlo algorithms are one way of sampling from a difficult or arbitrary distribution. MCMC methods simulate a Markov chain such that the stationary distribution of the chain is exactly the target distribution of interest. Fulfilling certain conditions ensure this property.

Cardinale (2013) presented a Markov chain design with an equilibrium distribution of $P(\Gamma|I)$. Where, simulating the Markov chain for a long time and storing all visited states, enables computing uncertainty statistics. We describe this design in subsection 2.4. First, we briefly review Markov chain theory in general.

A Markov chain is a random process that undergoes transitions from one state to another in a state space. The state space of the Markov chain defines the range of values that the random variables can assume. They can be either countable, have a discrete distribution, or be continuous. We only consider discrete state spaces. The set of discrete space variables may be finite or countably infinite (Ó Ruanaidh and Fitzgerald, 1996).

A Markov chain is defined as a series of random variables $X_0, X_1, \cdots X_n$ such that their influence on the future state (or value) X_{n+1} is only through the value of X_n:

Definition 1. *Consider $X = \{X_n : n \geqslant 0\}$ is a stochastic process on a countable set S. It is a* ***Markov Chain*** *if for any $i_0, \cdots, i_{n-1}, i, j \in S$ and for all integers $n \geqslant 0$*

$$P(X_{n+1} = j|X_n = i, \cdots, X_0 = i_0) = P(X_{n+1} = j|X_n = i) \tag{2.6a}$$

$$P(X_{n+1} = j|X_n = i) = p_{ij} \tag{2.6b}$$

where p_{ij} is the probability for the chain to move from state i to state j. This transition probability satisfy $\sum_{j\in S} p_{ij} = 1, i \in S$ and the matrix $\mathsf{P} = (p_{ij})$ is the transition matrix of the chain.

Definition 2. *Condition 2.6a called the* ***Markov property****, says that each state of a Markov chain depends on the previous state only.*

From equation 2.6b, we see that the transition probabilities do not depend on the time parameter n (homogeneous Markov chain). Here, we do not address the none-homogeneous case where they do.

Since we consider a countable state space, we can label the states by integers, such as $S = \{0, \cdots, m\}$. Under this labeling, the transition matrix takes the form

$$\mathsf{P} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0m} \\ p_{10} & p_{11} & \cdots & p_{1m} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m0} & p_{m1} & \cdots & p_{mm} \end{bmatrix}. \tag{2.7}$$

A Markov chain starts from an initial distribution $P\{X_0 = i_0\}$. From there, the distribution of all further states X_n on S with transition probabilities p_{ij} and for any $i_0, ..., i_n \in S$ and $n \geqslant 0$ is determined,

$$P\{X_0 = i_0, \cdots, X_n = i_n\} = P\{X_0 = i_0\} \cdot p_{i_0,i_1} \ \cdots \ p_{i_{n-1},i_n}, \tag{2.8}$$

which means, the probability the Markov chain traverses a path $i_0, \cdots i_n$ given by the multiplication $p_{i_0,i_1} \ \cdots \ p_{i_{n-1},i_n}$ of the probabilities of all transitions.

A Markov chain may visit some states infinitely often and visit others only a finite number of times. In order to classify states of the Markov chain X_n, we review some fundamental properties.

Definition 3. *The **hitting time** τ, is the number of steps for reaching state j for the first time in the Markov chain sequence and is defined as,*

$$\tau_j = min\{n \geqslant 1 : X_n = j\}, \quad j \in S. \tag{2.9}$$

The probability that the chain starting at i enters j for the first time at the nth step is,

$$f_{ij}^n = P_i\{\tau_j = n\}, \quad n \geqslant 1 \tag{2.10}$$

where, $f_{ij}^1 = p_{ij}, \ i, j \in S.$

The probability that a chain that begins at i ever hits j is,

$$f_{ij} = P_i\{\tau_j < \infty\} = \sum_{n=1}^{\infty} f_{ij}^n. \tag{2.11}$$

Definition 4. *A state i is* ***recurrent*** *if the chain returns to i with probability one ($f_{ii} = 1$). It is* ***positive recurrent*** *if $E_i[\tau_i] < \infty$ and it is* ***null recurrent*** *if $E_i[\tau_i] = \infty$. A state i is* ***transient*** *if it is not recurrent.*

Whether a state i is recurrent or transient depends on the number of times (N), the Markov chain X_n visits state i,

$$N_i = \sum_{n=0}^{\infty} \mathbf{1}(X_n = i). \tag{2.12}$$

State j is accessible from state i, ($i \to j$), if $p_{ij}^n > 0 \ for \ n \geqslant 1$ (i.e., $f_{ij} > 0$).

Two states i and j, communicate with each other ($i \leftrightarrow j$), if state j is accessible from state i and state i is accessible from state j.

Definition 5. *C is a set of states in S. Any set C, is said to be* ***irreducible****, if for any states $i, j \in C$, states i and j communicate with each other.*

The Markov chain X_n is irreducible if its entire state space S is irreducible.

Definition 6. *The states of the Markov chain can be revisited in a periodic fashion. The* ***period*** *d_i of a state i is the largest integer such that $p_{ii}^n > 0$ if and only if n is a multiple of d_i. In other words, d_i is the greatest common divisor of all n that satisfy $p_{ii}^n > 0$. State i is* ***aperiodic*** *if $d_i = 1$.*

If a Markov chain is irreducible, all states of the chain are either positive recurrent, null recurrent or transient, and all states have the same period.

Definition 7. *If a Markov chain is irreducible, and its states are positive recurrent and aperiodic, it is called* ***ergodic****.*

To characterize the long-term or limit behavior of ergodic Markov chains, we need to know whether the chain converges to a limiting distribution, independent of the starting distribution.

Definition 8. *A probability distribution* π *is a* ***stationary distribution*** *for the Markov chain if,*

$$\pi(j) = \sum_{i \in S} \pi(i) p_{ij}, \quad j \in S \tag{2.13}$$

and $\pi = \pi \mathsf{P}$ *is a matrix notation of this equation.*

A stationary process is also called "in equilibrium" or a steady-state process. If an initial distribution of a Markov chain is a stationary distribution, it remains stationary. So, a chain that has reached the stationary distribution remains stationary in all subsequent moves.

An ergodic Markov chain converges to a unique stationary distribution regardless of the initial state.

A positive recurrent Markov chain has a unique stationary distribution π that satisfies the balance equations $\pi = \pi \mathsf{P}$.

Using the samples from a stationary Markov chain, it is possible to estimate the expectation of $E[f(X)]$ ($\sim \bar{f}$), for any function of interest f.

Definition 9. *Let* $\{X_n\}_{n \geqslant 0}$ *be an irreducible positive recurrent Markov chain with the stationary distribution* π*. Then,*

$$\bar{f} = \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} f(X_k) \tag{2.14}$$

is called an ***ergodic average****.*

Convergence to the required expectation is ensured by the *ergodic theorem*.

Equation 2.14 shows that a Markov chain can be used to estimate the expectation of f, where the expectation is taken over it's stationary distribution. However, first, we need to build a Markov chain in a way that its stationary distribution is exactly our target distribution $\pi(.)$. So, we need to ensure the correct stationary distribution and prove convergence. The Metropolis-Hastings algorithm provides such a method.

2.3.1 THE METROPOLIS-HASTINGS ALGORITHM

Metropolis et al. (1953) developed an algorithm that constructs a transition distribution from a symmetric proposal distribution such that the stationary distribution is exactly the given desired target distribution. Later the algorithm was generalized by Hastings (1970) for non-symmetric proposal distributions and referred to as Metropolis-Hastings (MH) algorithm. The derivation of the Metropolis-Hastings algorithm starts with the condition of detailed balance.

Definition 10. *If $\pi(i)$ and $\pi(j)$ be some probability distributions that the Markov chain is in state i and j, respectively.* ***Detailed balance*** *requires that each transition $i \rightarrow j$ be reversible.*

$$\pi(i) \cdot p_{ij} = \pi(j) \cdot p_{ji} \tag{2.15}$$

furthermore, satisfying detailed balance guarantees that π is a stationary distribution of the chain.

Detailed balance is a sufficient, but not necessary, condition for equilibrium. This means that a Markov chain can also have π as a stationary distribution without satisfying Equation 2.15.

Hastings (1970) showed that sampling with transition probability α_{ij}

$$\alpha_{ij} = \min\left(\frac{\pi_j \cdot p_{ji}}{\pi_i \cdot p_{ij}},\ 1\right) \tag{2.16}$$

results in a Markov chain that satisfies the detailed balance condition. The constructed transition distribution subjects the newly proposed sample to an accept or reject step. The ratio $\frac{\pi_j \cdot p_{ji}}{\pi_i \cdot p_{ij}}$ is typically referred to as the *Hastings ratio*.

The MH algorithm, therefore, guarantees that the target distribution is a stationary distribution of the chain. Furthermore, Markov chain theory states that the resulting Markov chain is guaranteed to converge uniquely to the stationary distribution, if it is ergodic.

Algorithm 1: Metropolis-Hastings algorithm

```
Initialize $X_0$ and set $t = 0$ ;
repeat
    Set $X_t \leftarrow i$ and draw a state $j$ from $p_{i_t}$;
    Sample a uniform(0,1) random variable U ;
    if $U \leqslant \alpha_{ij} = \min\left(\frac{\pi_j \cdot p_{ji}}{\pi_i \cdot p_{ij}}, 1\right)$ then
        $X_{t+1} \leftarrow j$;
    else
        $X_{t+1} \leftarrow i$;
until convergence;
```

The fraction of accepted moves among all trials is called an *acceptance rate* (AR). The term $\frac{\pi_j}{\pi_i}$ is called the *posterior ratio* and $\frac{p_{ji}}{p_{ij}}$ is called *forward-backward ratio* (FBR). The MH algorithm allows sampling from an arbitrary distribution with some user-specified proposal distribution. Directly sampling from the target distribution is typically infeasible (which motivates the need for MCMC sampling), but this observation gives insight in designing good proposals. The choice of the proposal distribution will often greatly impact the burn-in time, which is the time it takes for the chain to reach its stationary distribution. In particular, the closer the proposal distribution is to the target distribution, the faster the convergence. From the MH ratio, we see that the decision to move only depends on a ratio, so we only need to query an unnormalized version of $\pi(\cdot)$.

Cardinale (2013) has designed such a Markov chain for image segmentation that is irreducible and aperiodic.

2.4 REVIEW OF DISCRETE REGION SAMPLING

In Discrete Region Competition, the solution is a (local) maximum-a-posteriori (MAP) estimate, obtained in an iterative process. In this approach, the segmentation is a local maximum of a posterior and might not be the best solution. The results can also be sensitive to the initialization or regularization parameters. Even if the algorithm finds a global optimum, there is no information about the sensitivity or robustness of the obtained segmentation.

Inspired by the work of Chang and Fisher III (2011), and motivated by the need to quantify the uncertainties of segmentation results, Cardinale (2013) presented an efficient Markov-Chain Monte Carlo (MCMC) sampling from the posterior distribution $P(\Gamma|I)$ induced by the energy. The distribution is not known explicitly, but given the image I and imaging model, it is possible to evaluate the posterior $P(\Gamma|I)$ of segmentation. By drawing multiple samples from the distribution, the goal is to find a collection of segmentations. Here the idea is to construct a stochastic process, a Markov Chain, whose visited states mimic samples from the target distribution and its equilibrium distribution is the posterior distribution $P(\Gamma|I)$. The stochastic nature of the MCMC sampling approach enables overcoming local probability maxima. The method is called Discrete Region Sampling (DRS) and using Equation 2.14 over visited states; we can compute the desired statistics.

The MH algorithm (Metropolis et al., 1953; Hastings, 1970) by construction, produces a Markov Chain that is reversible, aperiodic, and irreducible. The reversibility, aperiodicity, and irreducibility are sufficient conditions for converging to the correct equilibrium distribution (Smith and Roberts, 1993).

DRS performs many small steps with a relatively high acceptance rate and differs from Chang and Fisher III (2011), where they try to increase the step sizes by drawing and applying multiple samples per move. Small step becomes possible by taking advantage of efficiently computing the energy difference for small shape deformation.

Objects are closed regions, represented by a label image L. Label image (or label function) L maps a discrete space coordinate x to the region label currently assigned to that pixel (Cardinale et al., 2012). Applying a move means adding pixels, or removing them, or both to a region. This causes a transition from one state (segmentation) to another. In this approach, the proposal is a discrete probability distribution, enabling computation of the probability of selecting an individual point when calculating the forward-backward ratio.

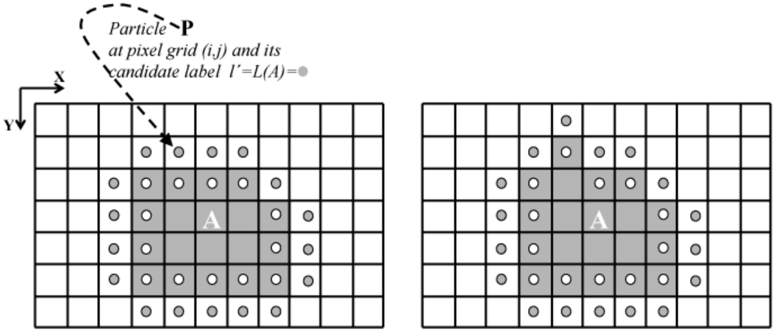


Figure 2.4: Illustration of a region A in a 2D digital image (grid). (a) Particles are shown as filled circles, with their colors indicating the candidate label. (b) Applying a move **p** *causes the label image to inherit* **p***'s candidate label at position (i,j) (Cardinale, 2013).*

Like in DRC, points are computational particles. Here, the particle properties are a candidate label and a weight. Particle weight is an unnormalized discrete probability. Weighted particles bias the proposal toward smooth contours. Without bias, every particle has the same weight as 1. To compute weight, we use length approximations for contours and surfaces introduced by Boykov and Kolmogorov (2003). The discrete probability mass function assigns a probability to each particle or set of particles (Cardinale, 2013).

Applying a move means the label image L inherits the particle's candidate label at that position. This is illustrated in Fig. 2.4.

To ensure reversibility, every move must have an associated reverse move to make a change back to the previous state possible. Figure. 2.5 illustrates the situation where applying particle $\mathbf{p}$ to move from state X_t to X_{t+1}, the reverse particle $\mathbf{p}'$ is not part of any region contour. The particle definition is hence augmented in this sense.

In order to distinguish the two types of particles, we call them *regular* and *floating* particles. Regular particles are bound to a contour (see Fig. 2.4) such that the label of the particle at position x is different from its candidate label, and it has at least one n-connected neighbor with label

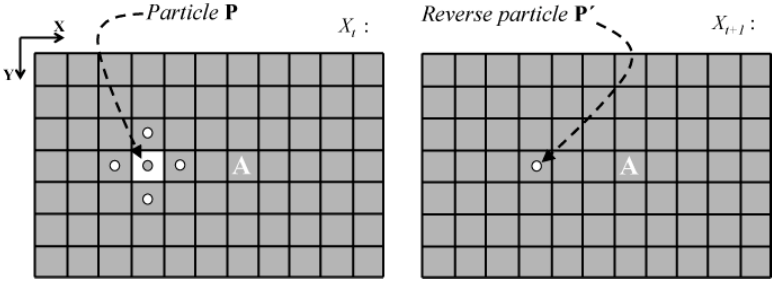


Figure 2.5: Illustration of a situation where applying particle $\mathbf{p}$ *and moving from state* X_t *to* X_{t+1}*, the reverse particle* $\mathbf{p}'$ *is not part of any region contour. The gray region is a foreground region and the background region is white.*

$l' \in [0; M]$ (the number of regions is fixed to M).

Any particle that is not regular is a floating particle. Figure 2.5 shows a transition from a regular to a floating particle as a hole is closed. A reverse change from a floating to a regular particle is also possible.

A joint set of regular and floating particles $(\mathcal{P} \cup \mathcal{P}_f)$ is used for the discrete proposal distribution.

Since the label image L does not contain floating particles, the state space S is augmented as,

$$S = \mathcal{L} \times \{\mathcal{P}_f\}, \tag{2.17}$$

where $\mathcal{L} = \{L_i\}$ is the collection of segmentations and $\{\mathcal{P}_f\}$ are all possible floating particle sets. This state space is huge. The size of S for M topologically unconstrained regions is

$$|S| = M^{|\Omega|} \cdot (|\Omega| - 1)^{M-1}, \tag{2.18}$$

where $|\Omega|$ is the number of pixels in the image.

The Markov chain moves from state X_t to X_{t+1} by either applying particles or changing the particle set, for example from floating to regular particles (Cardinale, 2013).

The algorithm starts from an initial label image L_0. L_0 is either the result of DRC or any initialization approach (frequently: local intensity maxima or an initial thresholding). In either case, L_0 defines the number of regions M, which is fixed from then on. L_0 and an empty set of floating particles represent the initial state of the Markov chain, X_0. $\mathcal{P}$ is the set of regular particles in the initial segmentation. Prior knowledge can be incorporated by biasing the proposal. For example, a smooth shape bias favors target distributions with smooth contours. Any bias can be incorporated through discrete proposals using particles weight (Cardinale, 2013).

Sampling a particle is efficiently done by splitting the particle set into subsets according to the region they belong. Sampling then chooses a region l from a uniform distribution, where every region has an equal probability of $\frac{1}{M-1}$. Note that any other splitting of the particle set can be used as well (Cardinale, 2013).

A particle is sampled either from the floating or regular particle sets. During the burn-in phase, the algorithm explores interesting regions away from the contour with probability q_f (*off-boundary sampling*) or with probability $1 - q_f$ from the whole set $\mathcal{P} \cup \mathcal{P}_f$. *Off-boundary sampling* samples over floating particles, applying or deleting a particle from $\mathcal{P}_f$. It explores parts of the image with large intensity gradients $|\nabla I|$ and can be disabled by setting $q_f = 0$.

After drawing a particle, we remove it from either $\mathcal{P}$ or $\mathcal{P}_f$ and insert the reverse particle in the appropriate set. Updating the particle weights in the neighborhood of the applied particle is accompanied by deleting all particles that do not satisfy the topological constraints.

For the new proposed state X', we can compute the backward proposal and energy difference $\Delta\mathcal{E}$ incurred by the particle move. From Markov chain theory, we can evaluate posterior ratios ($\frac{\pi'}{\pi} = \exp(-\Delta\mathcal{E})$). Now, it is possible to apply the MH algorithm to accept or reject the move (Cardinale, 2013).

A DRS approach is more robust to initialization and local optima than DRC and provides confidence bands for segmentations. But, it is computationally expensive, and the algorithm requires more memory than DRC. To address these problems we exploit parallel hardware and distributed computing.

2.5 Distributed computing

Based on the fundamental principle of dividing large problems into smaller ones, and solving them simultaneously with multiple resources, parallel computing, and distributed systems have been employed for many years.

A distributed computing system is a computer network in which components located on networked computers communicate and coordinate over the network to achieve a common goal. Solving computational problems using distributed systems in a general sense is called distributed computing. Distributed computing has been of increasing interest in the scientific community since the early 1980's. With the rapid improvement and availability of high-performance components, the power of distributed computing becomes more and more apparent. A tremendous amount of research has been done to overcome the intrinsic problems of distributed architectures and to perform robust computations ensuring high performance.

In distributed computing, each computing component has a private memory, and components exchange information by passing messages. Dividing a large problem into many parts, each component solves one part. The different components communicate with each other to ultimately solve the whole problem. Taking advantage of multiple resources, more memory and more computing power are available to address larger computational problems which otherwise are impossible or limited to the computing power and physical memory of a single source.

Although distributed computing is highly desirable, it is hard and challenging, since every component can only be aware of the information that it stores and has only a partial view of the entire problem.

We provide systematic solutions to difficulties and challenges in distributed parallelization of image segmentation and uncertainty quantification for large fluorescence-microscopy images in chapters 3 and 4.

In our approach, the parallel implementation is realized using the Parallel

Particle Mesh (PPM) library (Sbalzarini et al., 2006; Awile et al., 2010, 2013). This is natural due the particle-methods character of DRC and DRS.

2.5.1 Middleware for distributed computing: PPM

A primary focus in high-performance computing is the efficient execution of computer codes with the best utilization of a full range of software and hardware platforms. This requires significant intellectual and architectural investment and months or years of code development. One possible approach to alleviating this is to use the middleware that sits in between the applications and the distributed hardware platforms.

Since our methods of interest in image processing (segmentation and uncertainty quantification) are particle-mesh methods, we use the Parallel Particle Mesh (PPM) library (Sbalzarini et al., 2006; Awile et al., 2010, 2013) as a middleware for distribution and orchestration of the parallel communication.

PPM provides a processor-independent, data-transparent programming model for distributed-memory computers and hides the Message Passing Interface (MPI) from the application program. It introduces an additional, transparent layer between libraries such as MPI, METIS (Karypis and Kumar, 1998), or FFTW and an application program implementing an algorithm using particles, meshes, or a combination of the two, without losing generality concerning the models.

To further reduce code development times, a domain-specific programming language (DSL) for particle methods has been developed. This PPML (Awile et al., 2013; Awile, 2013) language exposes the PPM abstractions in an embedded DSL (eDSL), embedded in Fortran 2003. The PPML source-to-source translator converts PPML code to standard Fortran 2003 code that links against the PPM library as its runtime system.

In this thesis, for the first time, we extend the PPM to a non-simulation task, namely image analysis.

CHAPTER

THREE

Parallel distributed-memory Discrete Region Competition algorithm

3.1 Introduction

We parallelize the Discrete Region Competition (Cardinale et al., 2012) algorithm in a distributed-memory environment by applying a domain-decomposition approach to the image. Here, we describe the design and implementation of the parallel approach. Our approach is specifically designed to support parallel execution on heterogeneous clusters and parallel machines.

3.2 Data distribution by domain decomposition

The input image is decomposed into disjoint sub-images that are distributed to different computers. This is illustrated in Fig 3.1. Domain decomposition and data distribution are done transparently by the PPM

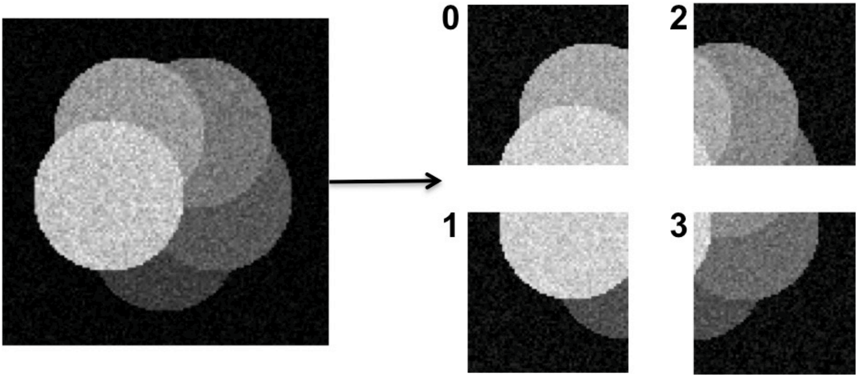


Figure 3.1: An illustrative example showing domain decomposition and distribution of an image across four computers (numbered 0 to 3).

library (Sbalzarini et al., 2006; Awile et al., 2010, 2013). Reading the input image from a file is also done in a distributed way, where each computer only reads certain image planes. The PPM library then automatically redistributes the data so as to achieve a good and balanced decomposition. Each computer only stores its local sub-image, and no computer needs to be able to store the entire image data.

3.2.1 Initialization

The algorithm is initialized locally on each computer, using only the local sub-image. The boundary information between sub-images is communicated between the respective computers with ghost layers. Ghost layers are extra layers of pixels around each sub-image that replicate data from the adjacent sub-images on the other processors, as illustrated in Fig. 3.2. The width of these ghost layers is determined by the number of pixels required to compute energy differences, i.e., by the radius of the energy kernel (see Ref. (Cardinale et al., 2012) for details). The width of the ghost layer defines the communication overhead and hence the parallel

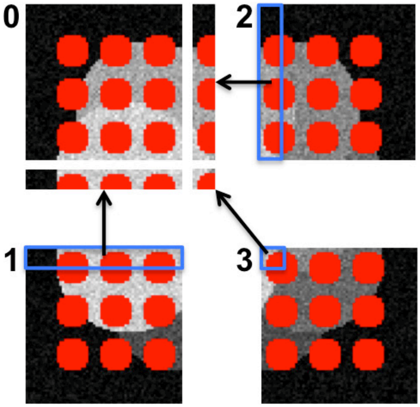


Figure 3.2: Ghost layers communicate information between neighboring sub-images residing on different computers. In the example from the previous figure, processor 0 receives ghost data from processors 1, 2, and 3, as shown for a ghost layer width of 10 pixels. The same is also done on all of the other processors. This allows the particles (boundary pixels of red regions) to smoothly migrate across computers, and energy differences to be evaluated purely locally on each sub-image.

scalability of the algorithm. PPM ghost mappings (Sbalzarini et al., 2006; Awile et al., 2010, 2013) are used to transparently update and manage ghost layer information whenever the corresponding pixels on the other computer have changed.

The initial segmentation from which the algorithm starts can be determined in a number of ways. Figure 3.2 shows an example of an initial segmentation given by uniformly distributed circles (shown in red). From there, the algorithm evolves to the final result. Using an initialization that is so far from the final result, however, increases the runtime and also bears the risk of converging to a sub-optimal local energy minimum. In practice, we hence usually initialize by a local-maximum detection or an initial intensity thresholding.

Starting from the initial segmentation, each FG region is identified by a globally unique label (Cardinale et al., 2012). This requires care in

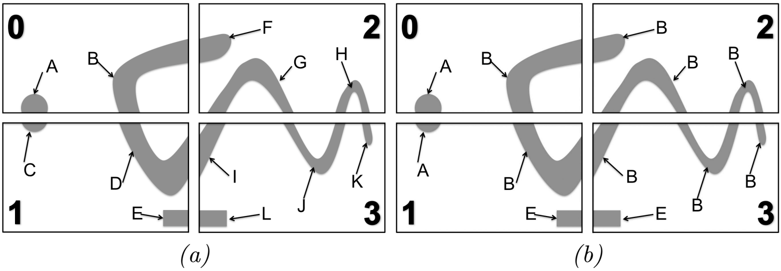


Figure 3.3: Region label initialization starts by each processor assigning unique labels to the FG regions in its sub-image (a). This, however, leads to conflicts for regions extending across multiple sub-images, as they will receive multiple, conflicting labels. Using a parallel connected-component algorithm (Flanigan and Tamayo, 1995), these conflicts are resolved in a second step, leading to a globally unique label for each FG region (b).

a data-distributed setting, since different computers could use the same label to denote different regions. In our implementation, each processor first performs an intermediate local labeling of the regions in its sub-image. Using the processor number (i.e., processor ID), this is done such that no two labels are used twice (see Fig. 3.3a). All regions are hence labeled uniquely. However, regions extending across more than one sub-image will be assigned multiple, conflicting labels. In a second step, the algorithm resolves these conflicts, ensuring that each FG region is uniquely labeled.

Using the definition that a FG region has to be a connected set of pixels, uniquely labeling them can be done using a parallel connected-component algorithm (Flanigan and Tamayo, 1995; Knop and Rego, 1998; Teuler and Gimel, 2000; Tiggemann, 2001; Wang et al., 2003; Moloney and Pruessner, 2003).

3.2.2 Parallel connected-component labeling

We implement and use the parallel connected-component labeling algorithm proposed by Flanigan and Tamayo (1995), which is based on an iterative relaxation process. During this, each sub-image exchanges boundary-crossing labels with neighboring processors. The labels are then replaced by the minimum of the two labels from the two processors. This process continues in iterations until a fixed point (labels do not change anymore on any processor) is reached (Flanigan and Tamayo, 1995). This is only done once, during the initialization, and leads to a result as illustrated in Fig. 3.3b. Every FG region now has a globally unique, unambiguous label, independent of which sub-image it lies in, or across how many computers the image has been distributed.

This iterative approach is efficient as long as the initial regions span only a few processors, as the blob-like structures of cell nuclei, where one or two iterations suffice to converge to the fixed point. If, however, an object spans across many sub-images, like a vascular network, a large number of iterations may be required.

The maximum number of iterations needed in this scheme for information to travel from one corner to the opposite corner of a slab domain-decomposition is $3 \times P^{\frac{1}{3}}$ (Teuler and Gimel, 2000), where P is the number of processors. For other decompositions, this can be even higher. Fig. 3.4 illustrates an example of an object that spans over all processors, along with the iterations required to reach a unique connected-component labeling. The diffusive nature of the method thus prevents fast initialization in cases where regions span many processors by requiring many iterations.

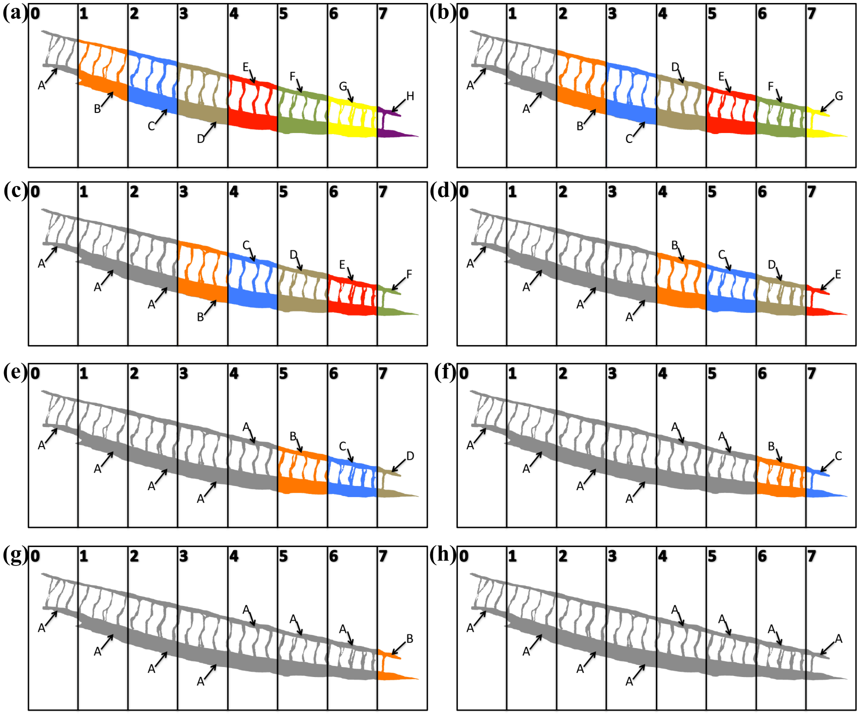


Figure 3.4: Distributed parallel connected-component labeling (Flanigan and Tamayo, 1995) on eight processors. Region label initialization starts by each processor assigning unique labels to the FG regions in its sub-image (a). Seven iterations are required in this example to converge to a globally unique label (b)-(h).

To address this problem, Moloney and Pruessner (2003) proposed a master-slave method for connected-component labeling on distributed-memory machines. The algorithm also starts by all slave processors independently labeling their local parts of the objects in parallel using the Hoshen-Kopelman algorithm (HKA) (Hoshen and Kopelman, 1976) (see Fig. 3.5).

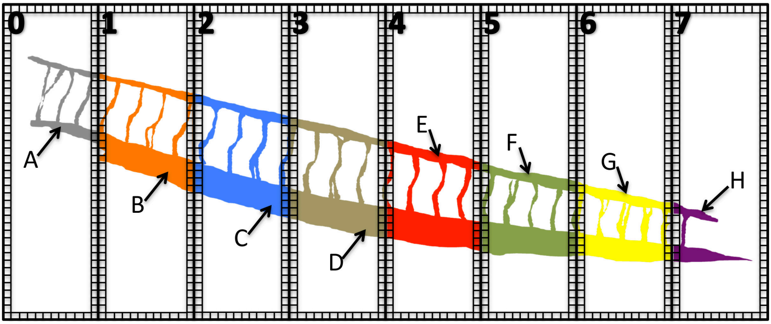
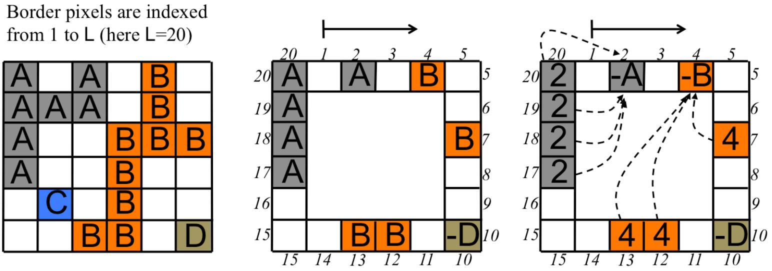


Figure 3.5: Every processor independently labels the local part. The border around each processor is pixelated for clarity and different labels are shown with different colors.

In the following step, the border pixels of each sub-image are indexed from 1 to L on each processor, where L is the local length of the border. For every component connected to the border, the algorithm assigns the first pixel in the border as the root of that component. This is done by traversing the border in succession from 1 to L. The first site of a previously unseen label is called the root and marked with a negative value. All other pixels of the same label in the border refer to the pixel index of the root. This is illustrated in Fig. 3.6. This relabeling allows identifying the root directly from every pixel of a region.



(a) (b)

Figure 3.6: (a) An example of region labels in one sub-image. (b) The border labels that will be sent to the master node. Border scan starts at border pixel one. Region roots are marked with negative labels and all other pixels of the same region in the border point to the position (border pixel index) of the root.

This border relabeling procedure is detailed in Algorithm 2.

Algorithm 2: Border relabeling

```
Create an empty hash map $\mathcal{M}$ ;
for $i \leftarrow 1$ to L do
    $value = M.find(\ |label(i)|\ );$
    if $value \neq NULL$ then                // label exists in $\mathcal{M}$
        $label(i) = value;$
    else                                     // label does not exists in $\mathcal{M}$
        if $label(i) \neq 0$ then
            $M.insert\ (\ label(i),\ i\ );$
            $label(i) = -label(i);$
```

After border relabeling, all processors send a representation of their borders to the master node. The master node combines them into an ordered arrangement (Moloney and Pruessner, 2003). To do this, the master node requires enough memory to hold all incoming borders, while the slaves only need to store local data.

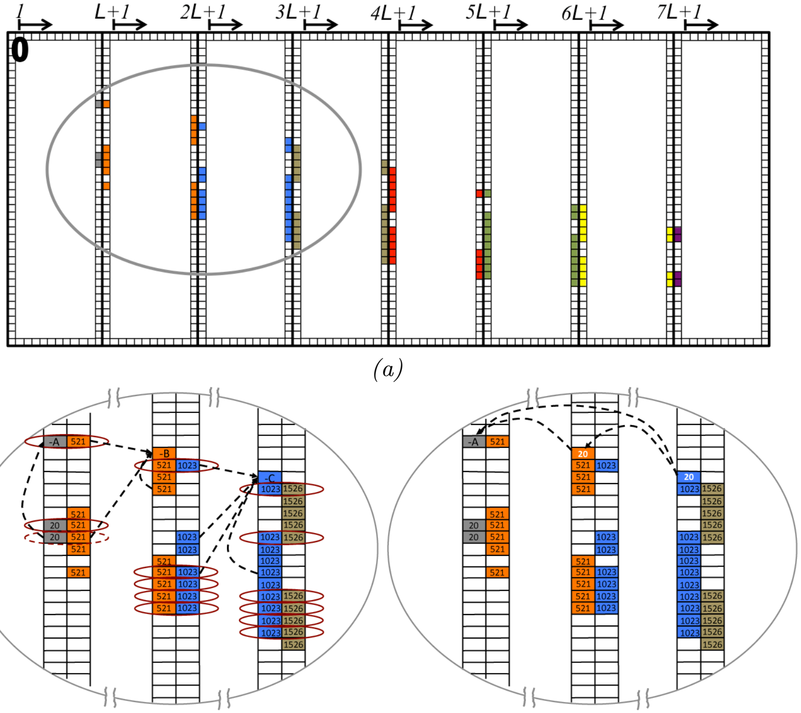
The master node combines the incoming borders by first concatenating them (see Fig. 3.7a). The pixel indices of the incoming borders are shifted by the size of the previous borders in order to render them globally unique. Figure 3.7a illustrates the configuration of the borders and the starting index of each one of them on the master processor after receiving and concatenating them from the other processors. In this example, the lengths of all borders are equal to L.

The master processor then iterates through all connected pixels from adjacent borders (red ellipses in Fig. 3.7b) and merges them. Merging is done by setting one of the roots (the one with the bigger label) to point to the other root, as shown in Fig. 3.7c.

In the example of Fig. 3.7b, the connected pixels marked with a dashed red ellipse. The lowest root ($-\mathsf{A}$) has index 20. At the other connected pixel, the value is 521, meaning they point to the root at index 521 with region label B. Since $\mathsf{A} < \mathsf{B}$, the merge happens by changing the root at pixel index 521 to point to pixel index 20 (i.e., change the value $-\mathsf{B}$ to 20).

This method is efficient and scales to large lattices. The authors reported results on a lattice of size $22.2 \cdot 10^6 \times 22.2 \cdot 10^6$ (Moloney and Pruessner, 2003). But, to find the connected components in the initialization step of our image-segmentation task and label them, especially for three-dimensional images, the serialization on the master processor would be a bottleneck.

We address this issue by modifying the algorithm, and introducing a new parallel connected-component labeling algorithm. In our approach, instead of gathering the pixel border information on a master processor, we further reduce and contract the results locally. This is made possible by the ghost layer, which we require anyway. On every processor, we thus create an array of unique pairs of connected components on the border using the ghost-layer information. This step is local and fast, as we only save the unique pairs in an array. This is illustrated in Fig. 3.8 for processor ID 2 from Fig. 3.5.



(a)

(b)

(c)

Figure 3.7: Every processor sends its border to the master node (Here processor with ID 0). (a) The configuration of borders on the master processor after receiving and concatenating them from the other processors. The pixel indices of the incoming borders are shifted by the size of previous borders to render them unique. (b) Red ellipses mark connected pixels from adjacent borders. A negative number is a component root at the border. A positive number is the pixel index of the root. (c) Merging is done by setting one of the roots of the connected pixels (the one with the bigger label) to point to the other root. Labels that have changed after the merging procedure are shown in white.

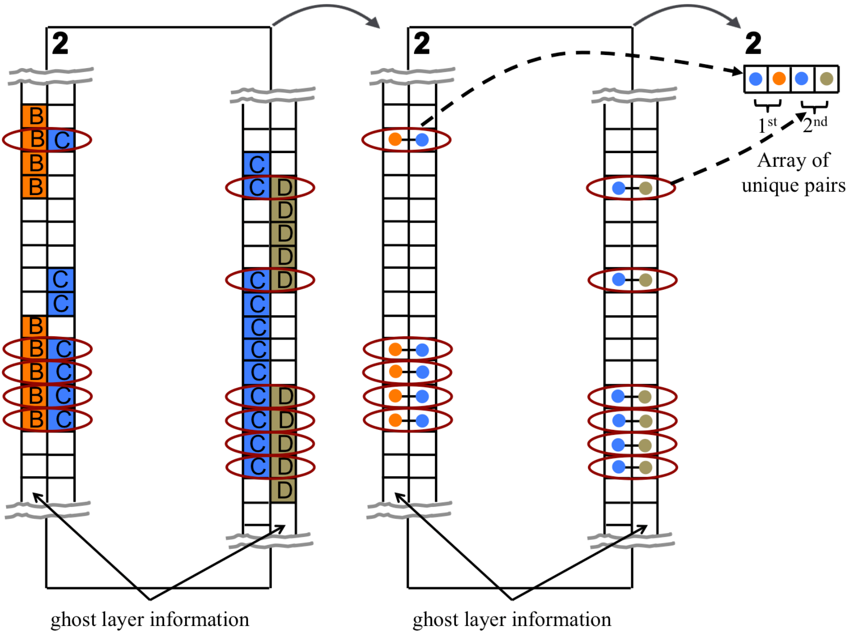


Figure 3.8: Part of the border, ghost layer, and pixel labels on processor ID 2. Red ellipses mark connected pixels. Colored circles connected by a line indicate connected components. An array of unique pairs of connected components is created by removing duplicates.

We then use bitwise operations on pairs of labels (i.e., two integer numbers) in order to combine them into one long integer:

```
long bothlabels = (long) firstlabel << 32 | secondlabel,
```

where `<<` is a left bit-shift and `|` is a bitwise inclusive-OR operation.

These long integers are then sorted into an array in ascending order, which in most cases enables finding any pair (one long integer) in $\mathcal{O}(1)$ time and at worst in logarithmic time (Knuth, 1998). Two `MPI_Allgather` operations are then sufficient for every processor to have a complete collection of unique label pairs. The first `MPI_Allgather` communicates the sizes of the arrays, in order to determine the total size of the global array. The second one gathers the array entries from all processors.

We use the global array of unique pairs to create an undirected graph locally on each processor, where each pair is an edge, and its two labels are the vertices of that edge.

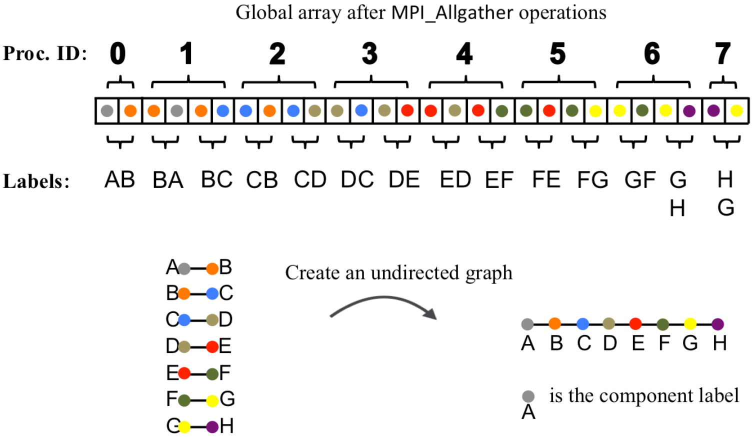


Figure 3.9: The global array of all unique pairs is used to create an undirected graph, where each pair is an edge, and its two labels are the vertices of that edge. The smallest label within each connected sub-graph is the new component label.

Connected sub-graphs of this graph then correspond to the connected components in the image. The smallest label within each connected sub-graph is the new component label. This is illustrated in Fig. 3.9.

Since this information is local to every processor, there is no need for further communication. Then, by traversing each connected sub-graph, for each vertex, we replace the corresponding label in the image with the new component label.

Traversing the global array and building the graph is fast, as the size of the array is limited to unique pairs across all processors. This approach hence accelerates the initialization of the algorithm for objects that span across many sub-images on large numbers of processors.

The algorithm only requires one iteration that involves two global communications. Our algorithm hence minimizes the amount of data exchange between processors and it is optimal in the number of communication rounds and in the amount of data transfer. The total number of communicated pairs equals the number of regions connected to or crossing any border. In our vasculature example of Fig. 3.4, there is only one unique pair per processor.

This benefit significantly shows on more than 32 processors, where the iterative approach performs considerably worse. The poor performance of the iterative approach is due to the number of iterations it requires. In our experience, the meantime of our algorithm for a constant problem size scales with $\mathcal{O}(\log P)$, where P is the number of processors.

3.2.3 PARALLEL CONTOUR PROPAGATION

Following initialization and initial region labeling, particles move across the image as driven by the energy-minimization flow in order to compute the segmentation. As outlined in section 2.2, this involves construction of a dependency graph of causally dependent particle moves, followed by selecting a maximal set of non-interfering moves. In a data-distributed setting, the problem occurs that every computer only knows the part of the graph that resides in its local sub-image. If graphs span across processor boundaries, correct move selection cannot be guaranteed without additional communication. This communication between computers is required to find a globally consistent set of independent moves, but should be kept to a minimum in order to guarantee algorithm scalability.

In our implementation, finding a globally consistent set of moves starts by each processor P creating a local undirected graph G_P, comprising its interior and ghost particles. Disconnected parts of the graph that entirely lie within the local sub-image are called *interior sub-graphs* G_P^i. Parts of the graph that extend across sub-image boundaries are called *boundary sub-graphs* G_P^b.

Identifying compatible moves in an interior sub-graph can be done independently by each processor. Resolving boundary sub-graphs, however, requires information from all sub-images across which the sub-graph extends. This is challenging because the sub-graphs sizes, structures, and distributions are not known *a priori*, as they depend on the input image.

Traditionally, master-slave approaches have been used to solve this problem on distributed machines. This is illustrated in Fig. 3.10. In this approach, the boundary sub-graphs G_P^b from all processors are gathered on one single processor, the master processor. This master processor then determines the move sets and sends them back to the respective other processors. Meanwhile, the other processors work on their interior sub-graphs.

This approach is easy to implement, but carries substantial overhead due to the global communication and the task serialization on the master processor. As we later show (Fig. 3.26), this approach does not scale and prevents acquisition-rate image analysis.

We address this problem by introducing a new parallel contour propagation algorithm, which does not require global communication and incurs no serialization. In theory, it hence scales perfectly. Instead of gathering all boundary sub-graphs G_P^b on one master processor, we propose to use the locally available boundary sub-graph on each processor and identify the compatible moves only on that local part. If all processors did this in parallel, however, conflicting moves across sub-image boundaries would occur. We avoid this by decomposing the processors into two sets:

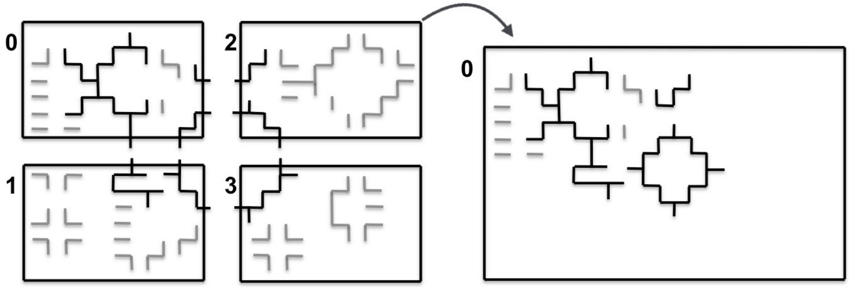


Figure 3.10: The master-slave approach to finding the global independent move set by gathering all boundary sub-graphs on a single master processor and then sending back the results. In this example, processor 0 is the master.

black and white processors. Since FG regions are face-connected, using a checkerboard decomposition as illustrated in Fig. 3.11 ensures that boundary sub-graphs always cross from black to white processors, or vice versa. They never cross sub-image boundaries within one color, hence avoiding boundary conflicts if the processing is done by color. Therefore, the black processors start by determining the viable moves on their boundary sub-graphs, while the white processors work on their interior sub-graphs. Then, the black processors communicate their boundary decisions to the neighboring white processors using a ghost particle mapping (Sbalzarini et al., 2006). Finally, the white processors resolve their boundary sub-graphs using the received decisions as boundary conditions, while the black processors work on their interior sub-graphs. This procedure is illustrated in Fig. 3.11. It effectively avoids conflicts and determines a globally viable move set within two rounds of local ghost communication.

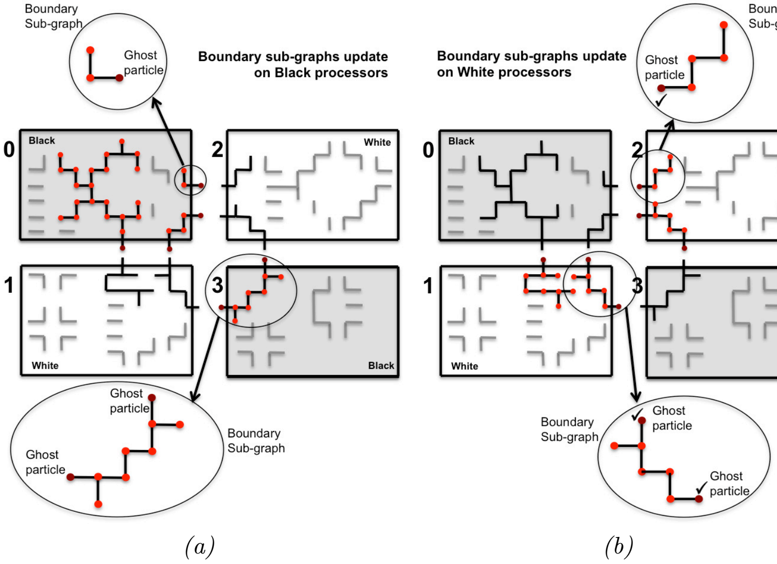


Figure 3.11: Distributed sub-graph algorithm to determine a globally consistent set of particle moves. The processors are divided into black and white ones using a checkerboard decomposition. (a) Compatible moves are identified simultaneously on all boundary sub-graphs (black) on the black processors, while the white processors work on their interior sub-graphs (gray). (b) Boundary particles are send from the black to the white processors in order to provide the boundary condition for the boundary sub-graph processing on the white processors. The ghosts are not altered by the white processors, but immediately accepted as moves (symbolized by the check marks). This avoids conflicts and only requires local communication.

Taking advantage of non-blocking MPI operations, the whole procedure is executed in an asynchronous parallel way, as detailed in Algorithm 3. This largely hides the communication time of the ghost mappings, resulting in better scalability and speed-up on a distributed memory parallel machine.

Algorithm 3: Parallel distributed-memory contour propagation algorithm

```
Find: interior and boundary maximal connected sub-graphs, $G_P^i$ and $G_P^b$
if Black processor then
    Receive: ghost information from white neighbor processors

    foreach boundary sub-graph $G_P^b$ do
        identify compatible moves

    Send: non-blocking send of updated boundary particle
          information to white neighbor processors

    foreach interior sub-graphs $G_P^i$ do
        identify compatible moves

    Wait: for non-blocking send to complete

if White processor then

    Send: non-blocking send of boundary particle information to
          black neighbor processors

    Receive: non-blocking receive of ghost information from black
             neighbor processors

    foreach interior sub-graph $G_P^i$ do
        identify compatible moves

    Wait: for non-blocking receive to complete

    foreach boundary sub-graphs $G_P^b$ do
        identify compatible moves under ghost constraints
```

Breaking the boundary sub-graphs along sub-image boundaries changes the sorting order of compatible moves, and hence the convergence trace of the algorithm in energy space. Enforcing boundary conditions at the break points of the boundary sub-graphs amounts to an approximation of the original problem. This approximation is not guaranteed to determine the same global move set as the sequential approach, because the moves are only sorted by energy locally in each sub-graph, and not globally in the entire graph. However, as long as the statistical distribution of break points in the graph is uniform, the optimizer is still guaranteed to converge, albeit the path of convergence may differ. This is a famous result from Monte Carlo (MC) approaches to the Ising model (Pawley et al., 1985), where it has been shown that unbiased randomization of the moves may even accelerate convergence toward an energy minimum. In our case, the distribution of break points is indeed unbiased. This is because it is the result of a domain decomposition that depends on the number of processors used, and the graphs depend on the unpredictable image content. Therefore, independent unbiased breaking is satisfied, and the distributed approach converges.

We empirically confirm this convergence by comparing the energy evolution of the original sequential algorithm (Cardinale et al., 2012) and our new distributed method on different 2D benchmark images from the Berkeley database (Martin et al., 2001). The result for four example images is shown in Fig. 3.12 using different numbers of processors and hence different subgraph decompositions. In all tested cases, both methods converge. The largest observed difference in final energy is less than 0.5%.

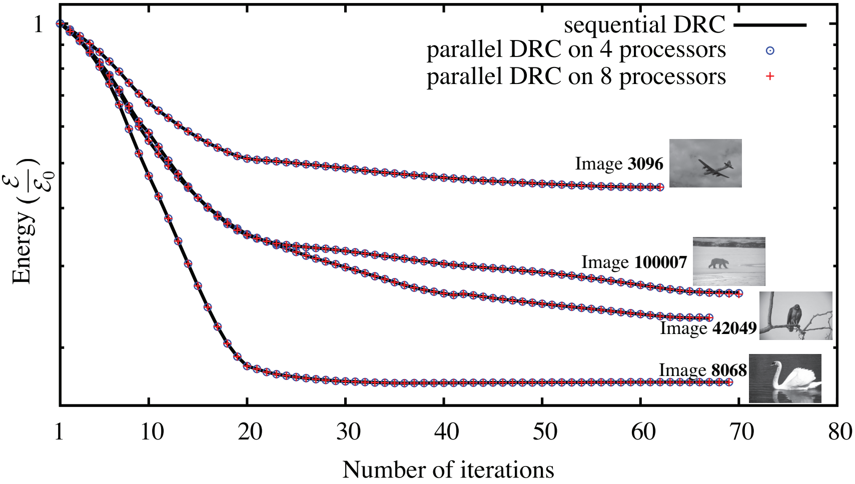
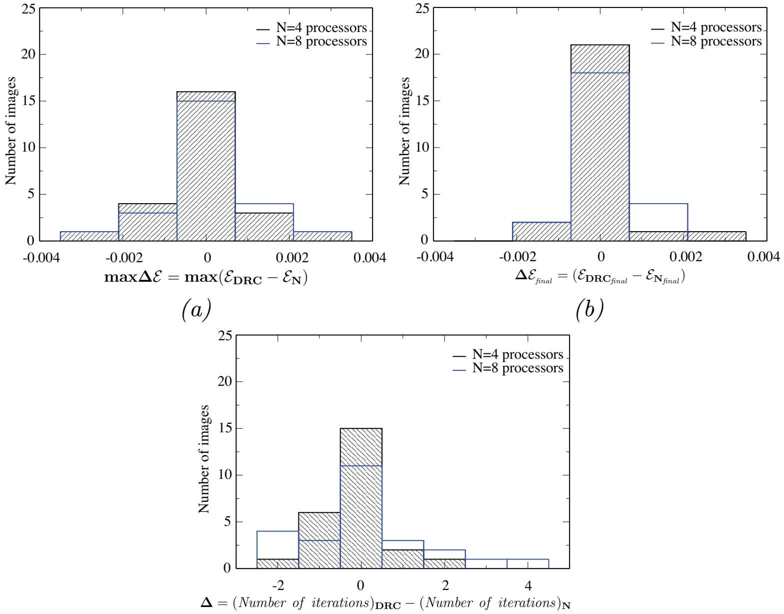


Figure 3.12: Energy evolution of the sequential DRC algorithm (Cardinale et al., 2012) and the present parallel algorithm on four different images from the Berkeley database (Martin et al., 2001) on 4 and 8 processors. Despite the boundary sub-graph decomposition (see main text), the results are pixel-wise identical in all cases except for image 100007, where two pixels on 4 processors and 3 pixels on 8 processors differ from the sequential result due to contour oscillations, as discussed in the main text.

Figure 3.13 shows histograms of the energy differences for 25 images from the Berkeley database. Three metrics are shown: (a) the maximum energy difference occurring anywhere along the convergence path, (b) the difference in the energy of the final converged state, (c) the difference in the number of iterations requires to reach convergence.



(a) *(b)*

(c)

Figure 3.13: Comparison of the results from the distributed DRC algorithm and the original sequential implementation (Cardinale et al., 2012) on 25 2D images from the Berkeley benchmark database (Martin et al., 2001). (a) Histogram of the maximum energy difference occurring anywhere along the energy evolution path; $\mathcal{E}_{DRC}$ is the sequential algorithm and $\mathcal{E}_N$ the distributed algorithm on N computers. (b) Histogram of the final energy difference of the converged solutions. (c) Histogram of the difference in the total number of iterations required to reach the converged final solution.

The results in Figs. 3.12 and 3.13 show that the parallel algorithm is in good agreement with the original sequential algorithm (Cardinale et al., 2012). Both algorithms show the same energy-evolution trend and converge to almost the same energy level with less than 0.5% difference anywhere during energy evolution. Figure 3.13c also confirms the Ising-model result that the randomized parallel algorithm on average converges in fewer iterations than the sequential method.

The question arises, however, if these small energy differences are significant in terms of the final segmentation. While no general guarantee can be given, the final segmentations were close in all cases tested, with at most 3 pixels differing between the sequential and parallel solutions. All observed pixel differences stemmed from contour oscillations around the converged state, as confirmed by pixel-wise comparison of the final segmentations. These oscillations are an inherent property of the energy descent method used in DRC (Cardinale et al., 2012). They are suppressed by reducing the number of concurrently accepted moves whenever oscillations occur (Cardinale et al., 2012). This is required in order to guarantee convergence of DRC. In our distributed DRC implementation, oscillations are detected locally by each computer. Also, the number of accepted moves per iteration is set locally, and potentially differently, by each computer. The oscillation pattern close to the final converged state is hence different than the one in sequential DRC. An example is shown in Fig. 3.14 where the only differences between the two segmentations are the two oscillatory particles shown in white. Since they may stop their oscillations at different locations, the final segmentations may differ in these two pixels, which explains the small energy difference. The final segmentations are, however, geometrically close, and the algorithm converges toward the same local energy minimum. It is also important to keep in mind that the sequential DRC algorithm uses an approximate local optimizer that may not find the globally best segmentation. Sometimes, the slightly different result obtained by the distributed method is therefore better in terms of energy (see Fig. 3.13b).

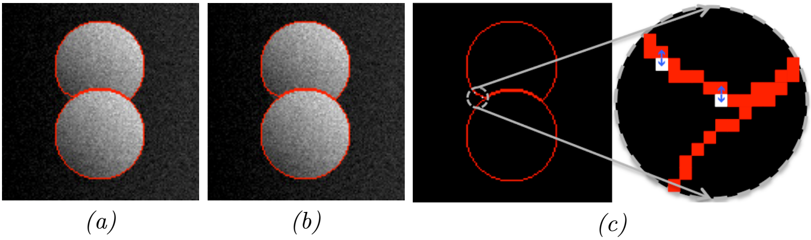


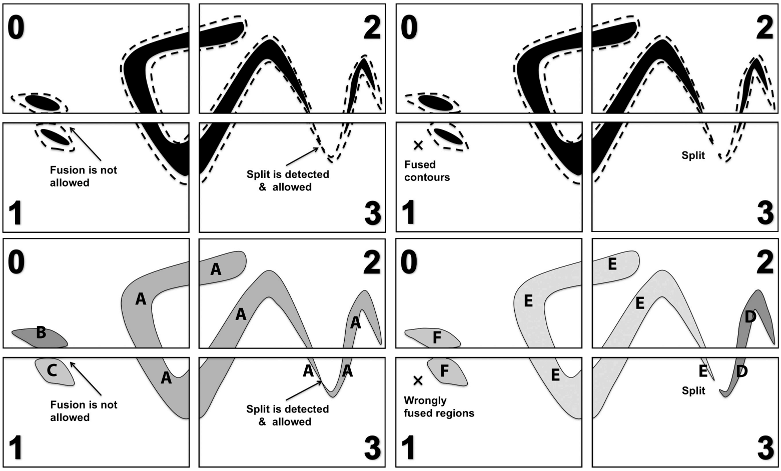
Figure 3.14: The small energy differences between the distributed and the sequential DRC implementations result from local pixel oscillations. An example is shown with a synthetic image using a piecewise smooth image model for segmentation. (a) Result on a single computer. (b) Result from the distributed algorithm on 8 computers. (c) Overlay of the two results with differences shown in white. These are two oscillatory particles jumping back and forth between two neighboring pixels. The final segmentation results are hence very close and amount to alternative pixelations of the object border line.

3.2.4 Parallel topology processing and data-structure update

After having determined the set of compatible acceptable moves, the particles (and hence the contours) propagate in parallel on each processor. This changes the region labels of the corresponding pixels, as regions move, shrink, or grow. Particles that move across sub-image boundaries are communicated to the respective destination processor using the local neighborhood mappings of PPM (Sbalzarini et al., 2006). This ensures global consensus about the propagating contours.

In addition to propagating, contours can also split or fuse if that is energetically favorable. This corresponds to a region splitting into two, or two regions merging. While digital topology allows such topological changes in the segmentation to be efficiently detected using only local information (Cardinale et al., 2012), the labels of the involved regions may change across sub-image boundaries. Whenever region labels change as a result of a split or fusion, a seeded flood-fill is performed in the original DRC algorithm (Cardinale et al., 2012), in order to identify the new connected components. This again requires additional care in a distributed setting, as illustrated in Fig. 3.15.

Figure 3.15a shows the two critical situations: two regions touching at a sub-image boundary that are not supposed to fuse (by the image energy model) and a split in a region that extends across multiple sub-images. The parallel connected component algorithm (see section 3.2.2) used during initialization would unnecessarily re-label all regions that cross any sub-image boundary and erroneously fuse the two touching regions (Fig. 3.15b). In order to obtain the correct result, we propose a particle-based update method, as detailed in Algorithm 4.



(a) *(b)*

*Figure 3.15: Distributed region split and merge algorithm. In the upper row, the evolving contours are shown by dashed lines and the underlying objects to be segmented by the black solid regions. (a) The situation before re-labeling the regions. Two regions (*B/C*) touch at a sub-image boundary, but should not fuse according to the image energy model. The region* A *extends across multiple sub-images and splits in sub-image 3. (b) Applying a parallel connected-component algorithm would erroneously fuse regions touching at sub-image boundaries and unnecessarily re-label all regions with new unique labels.*

Taking advantage of the particle representation of the evolving contours, information about region label changes at sub-image boundaries is communicated through PPM's ghost-get mappings (Sbalzarini et al., 2006). For each region, however, only two particles are communicated instead of the full ghost layer of pixels. This is illustrated in Fig. 3.16. Moreover, this only happens when the region label on the source processor did actually change. In this case, the corresponding boundary particles are activated. By default, all boundary particles are deactivated. Activated particles, which we call "hot particles" from now on, as inspired by the classical forest-fire algorithm, are then sent to the neighboring processor.

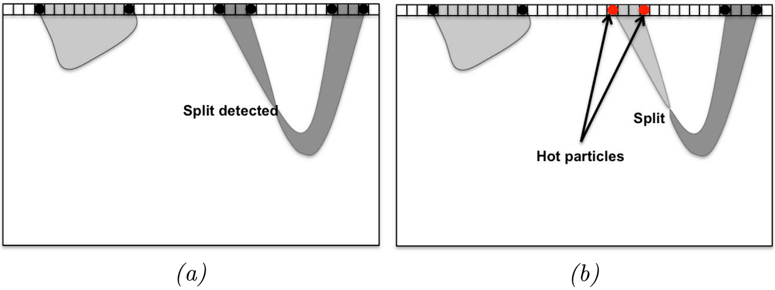


Figure 3.16: Boundary particles are activated upon region label changes in the local sub-image. Only activated ("hot") boundary particles are communicated to the neighboring processor, restricting re-labeling to affected regions and avoiding communication of a full ghost layer of pixels. (a) All boundary particles (black disks) are deactivated ("cold") before local region label update. (b) Boundary particles of re-labeled regions are activated (red disks, "hot") and propagate the label change to the neighboring processor.

The neighboring processor receiving the hot ghost particles starts a local forest-fire algorithm for seeded flood filling of the region, using the hot ghosts as seeds. Since this may propagate the label change across multiple processors, the procedure proceeds in iterations until no more hot particles are detected anywhere. This is determined by a global `MPI_Allreduce` operation of local flags for the presence of hot particles in each sub-image. Regions are always re-labeled using the lower of the two labels. This means that hot particles only propagate changes with new labels lower than existing ones. Therefore, the procedure is guaranteed to terminate, as oscillations or loops cannot occur.

The complete procedure is detailed in Algorithm 4. Again, all communication (mappings) is done asynchronously using non-blocking `MPI` operations. After execution of the algorithm, all regions are again identified by globally unique labels, but only necessary changes due to region splits/fusions have been made. Regions that did not undergo topological changes retain their previous labels. This prevents spurious region fusions and keeps the data-structure updates to a minimum. Also, communicating only hot ghost particles, instead of full pixel layers, significantly reduces the communication overhead.

Algorithm 4: Distributed region re-labeling

```
Reconstruct: label image $L$

$hotpart \leftarrow false$

Create an empty list $S$

if contour particle label changes at sub-image boundary then
    activate particle as a hot particle

    $hotpart \leftarrow true$

Ghost mappings: Particle

if there is any hot ghost particle then
    add it as a seed to the list $S$

Global: reduce operation on hotpart

while hotpart do
    Reconstruct label image $L$ using flood fill from the seeds in $S$

    Empty: $S$

    Ghost mappings: Particle

    if there is any hot ghost particle then
        add it as a seed to the list $S$

    $hotpart \leftarrow false$

    if contour particle label changes at particle ghost layer then
        activate particle as a hot particle

        $hotpart \leftarrow true$

    Global: reduce operation on hotpart
```

3.3 Results

We first show correctness and efficiency of the distributed parallel algorithm and then illustrate its application to acquisition-rate image segmentation in 3D light-sheet fluorescence microscopy. We demonstrate correctness by comparison with the original reference implementation of Cardinale et al. (2012) on synthetic and real-world images taken from the original DRC paper (Cardinale et al., 2012). Then, we assess the parallel efficiency and scalability of the present implementation using scalable synthetic images in both a weak-scaling and a strong-scaling experiment.

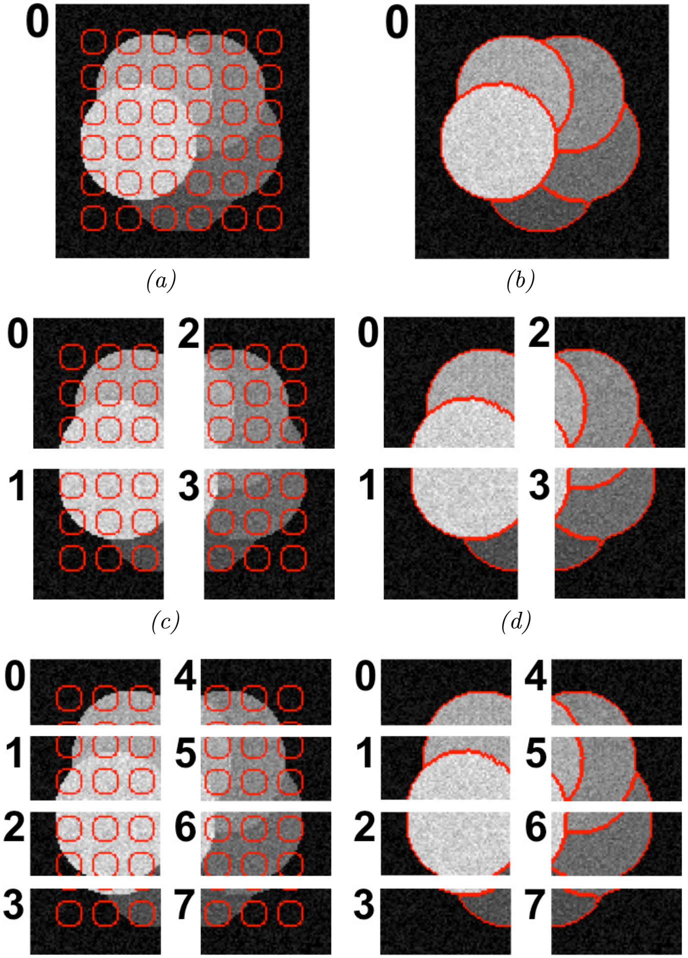
All computations were performed using the PPM library (Sbalzarini et al., 2006; Awile et al., 2010, 2013) in its 2015 version on the Bull cluster "Taurus" at the Center for Information Services and High Performance Computing (ZIH) of TU Dresden. The cluster island used consists of 612 Intel Haswell nodes with 24 cores per node and 2.5 GB of main memory per core. The parameter settings for all test cases are summarized in Table 3.1. They were determined following the guidelines given in the original DRC publication (Cardinale et al., 2012).

Table 3.1: Parameter settings used for the cases shown in this chapter (PC: piecewise constant; PS: piecewise smooth). See Cardinale et al. (2012) for parameter meaning and guidelines.

Initialization	Algorithm parameters	$\mathcal{E}_{\text{data}}$	Energy parameters
Icecream PC 2D, 130 × 130, Fig. 3.17			
6 × 6 bubbles	$\theta = 0.02, R_\kappa = 4$	PC	$\lambda = 0.04$
Bird, 481 × 32, Fig. 3.19			
32 × 21 bubbles	$\theta = 4.5, R_\kappa = 5$	PC	$\lambda = 0.2$
Cell nuclei, 672 × 512, Fig. 3.20			
local maxima	$\theta = 0.02, R_\kappa = 4$	PC	$\lambda = 0.06$
Icecream PS 2D, 130 × 130, Fig. 3.21			
5 × 5 bubbles	$\theta = 0.2, R_\kappa = 4$	PS	$\lambda = 0.04, \beta = 0.05, R = 8$
Elephants 2D, 481 × 321, Fig. 3.23			
21 × 14 bubbles	$\theta = 0.2, R_\kappa = 8$	PS	$\lambda = 0.2, \beta = 0.05, R = 4$
Zebrafish embryo germ cells 3D, 188 × 165 × 30, Fig. 3.24			
bounding box	$R_\kappa = 4$	PS	$\lambda = 0.08, \beta = 0.005, R = 9$
Synthetic unit cell test image 3D, 256 × 256 × 256, Fig. 3.25			
local maxima	$\theta = 0.02, R_\kappa = 4$	PC	$\lambda = 0.04$
Drosophila embryo 3D, 1824 × 834 × 809, Fig. 3.28			
local maxima from blob detector	$\theta = 0.001, R_\kappa = 8$	PC	$\lambda = 0.005$
Drosophila embryo 3D, 1824 × 834 × 809, Fig. 3.29			
local maxima from blob detector	$\theta = 0.001, R_\kappa = 8$	PS	$\lambda = 0.005, \beta = 1.0, R = 8$
zebrafish vasculature 3D, 1626 × 988 × 219, Fig. 3.30			
Li thresholding	$\theta = 10.0, R_\kappa = 8$	PS	$\lambda = 0.02, \beta = 0.001, R = 12$

3.3.1 Correctness of the distributed algorithm

Results using a piecewise constant image model. We first check that the distributed algorithm produces the same results as the sequential benchmark implementation in the case of a multi-region piecewise constant (PC) image model. In this model, the assumption is that different FG regions have different intensities that are, however, spatially constant within each region. We use the same test image as Cardinale et al. (2012) in order to compare the results. The result from the present distributed implementation is shown in Fig. 3.17 using one, four, and eight processors. Pixel-wise comparison shows that all segmentation results are identical to the one reported by Cardinale et al. (2012) for the sequential benchmark implementation.



(a) *(b)*

(c) *(d)*

(e) *(f)*

Figure 3.17: Distributed segmentation of a synthetic test image using a piecewise constant image model. (a) Initialization on a single processor with particles shown in red. (b) Final result on a single processor. (c) Initialization on four processors. (d) Final result on four processors. (e) Initialization on eight processors. (f) Final result on eight processors. The results are identical to those by Cardinale et al. (2012) in all cases.

Figure 3.18 shows the energy evolution of the distributed cases compared with the original sequential benchmark implementation of Cardinale et al. (2012). If anything, the graph randomization used in the present distributed algorithm slightly accelerates convergence in the first half of the iterations. All methods reach the same final energy.

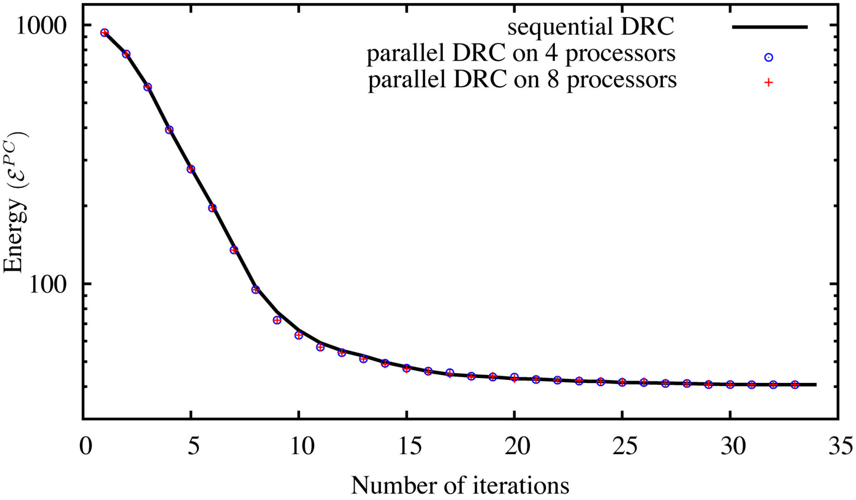
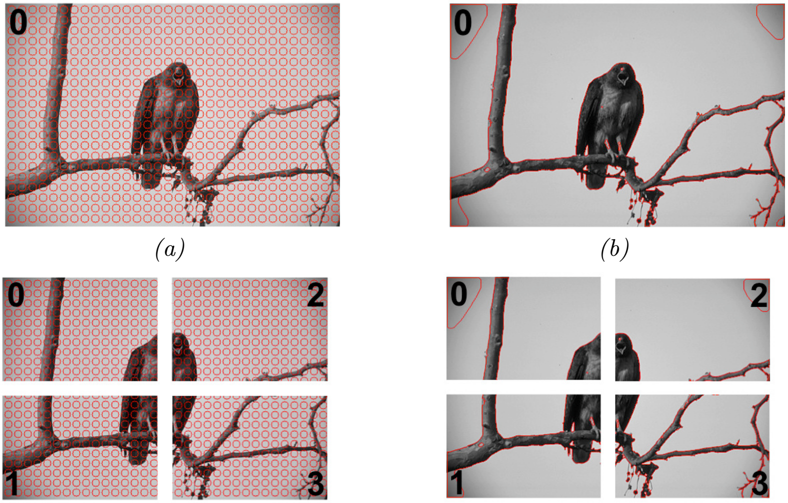


Figure 3.18: Energy evolution of the sequential DRC algorithm of Cardinale et al. (2012) in comparison with the present distributed algorithm processing the piecewise constant test image from Fig. 3.17 on four and eight processors.

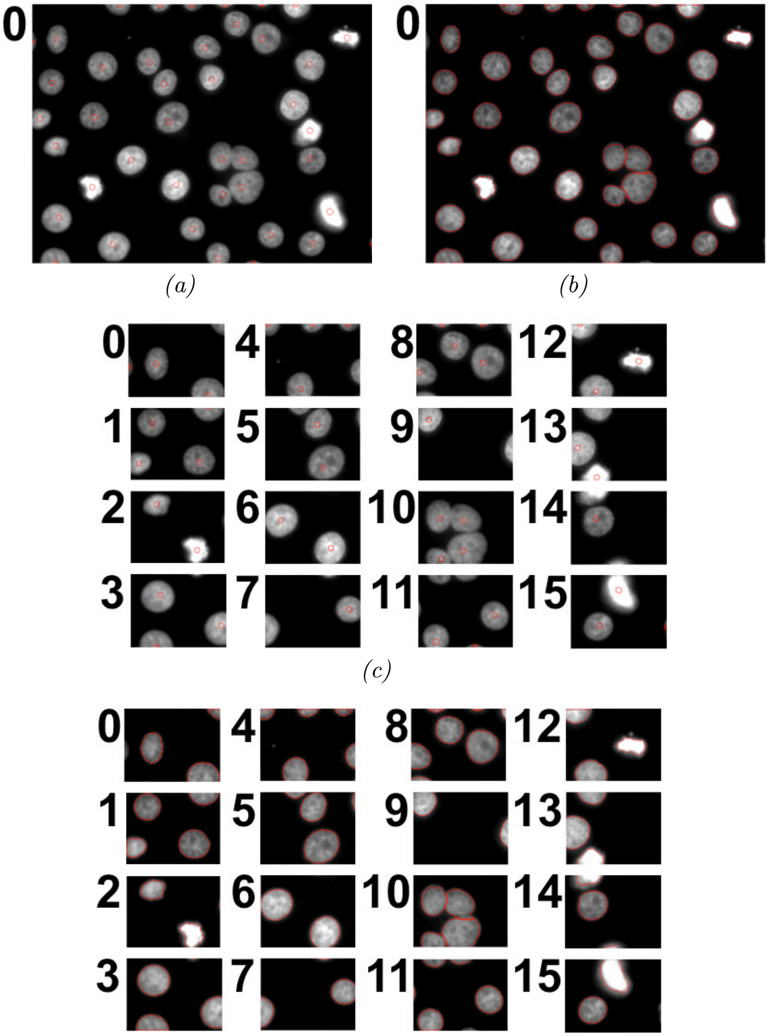
We further compare the results from the present distributed implementation with the original sequential algorithm on real image data. The same natural-scene image as considered in the original publication (Cardinale et al., 2012) is shown in Fig. 3.19. Again, the present implementation running on one and four processors produces the exact same result as the benchmark implementation.



(a) *(b)* *(c)* *(d)*

Figure 3.19: Distributed segmentation of a natural-scene image using a piecewise constant image model. (a) Initialization on a single processor with particles shown in red. (b) Final result on a single processor. (c) Initialization on four processors. (d) Final result on four processors. The results are identical to those by Cardinale et al. (2012) in both cases.

As a second real image, we consider the same fluorescence microscopy image of nuclei as in the original publication (Cardinale et al., 2012). The results on one and 16 processors are shown in Fig. 3.20. The algorithm is initialized with a circular region around each local intensity maximum after blurring the image with a Gaussian filter of width $\sigma = 10$ pixel. This is the same initialization as used by Cardinale et al. (2012). The results are identical, pixel by pixel.



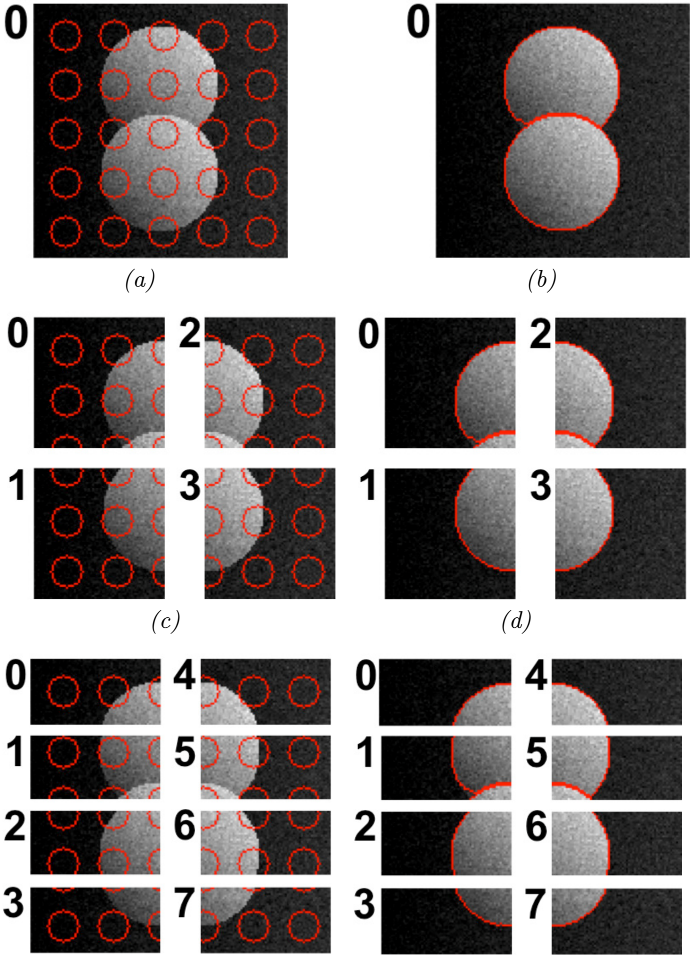
(a) (b)

(c)

(d)

Figure 3.20: Distributed segmentation of fluorescently labeled cell nuclei (raw image: Dr. Prisca Liberali, FMI Basel) using a piecewise constant image model. (a) Initialization on a single processor. (b) Result on a single processor. (c) Initialization on 16 processors. (d) Result on 16 processors. The results are identical to those by Cardinale et al. (2012) in both cases.

Results using a piecewise smooth image model. The DRC algorithm is generic over a wide range of image models, including the more complex piecewise smooth (PS) model. In this model, each region is allowed to have a smooth internal intensity shading. We again use the same synthetic test image as Cardinale et al. (2012) and illustrate the result in Fig. 3.21. Pixel-to-pixel comparison of the final segmentation results shows differences in two oscillatory pixels on eight processors (see also Fig. 3.14). This is consistent with the way boundary oscillations are detected and handled in the distributed algorithm in comparison with the sequential one.



(a) *(b)* *(c)* *(d)* *(e)* *(f)*

Figure 3.21: Parallel segmentation of a synthetic image using a piecewise smooth image model. (a) Initialization on a single processor with particles shown in red. (b) Final result on a single processor. (c) Initialization on four processors. (d) Final result on four processors. (e) Initialization on eight processors. (f) Final result on eight processors. Two oscillatory pixels differ with respect to the result of Cardinale et al. (2012) (see also Fig. 3.14).

The energy evolution for this case is shown in Fig. 3.22 in comparison with the original sequential DRC algorithm of Cardinale et al. (2012). Again, the two convergence traces are almost identical with small differences stemming from the graph decomposition used in the present implementation. The difference in final energy is due to the two oscillatory pixels, as discussed above and shown in Fig. 3.14.

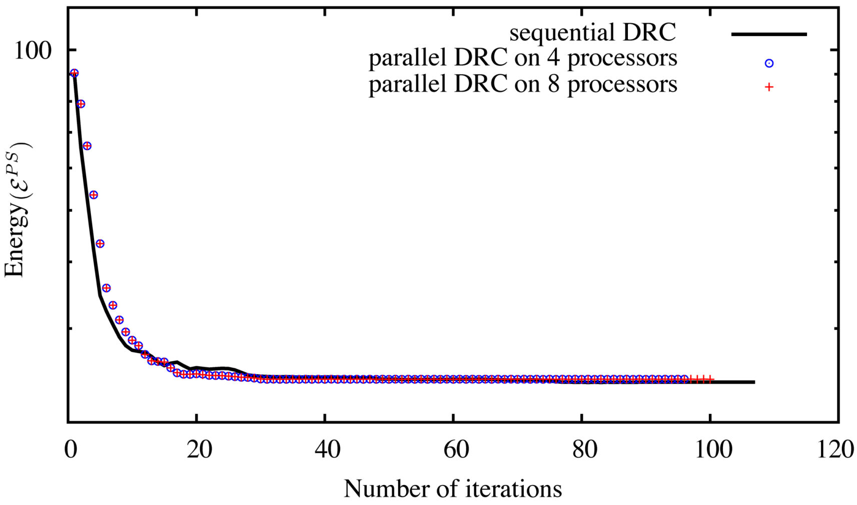
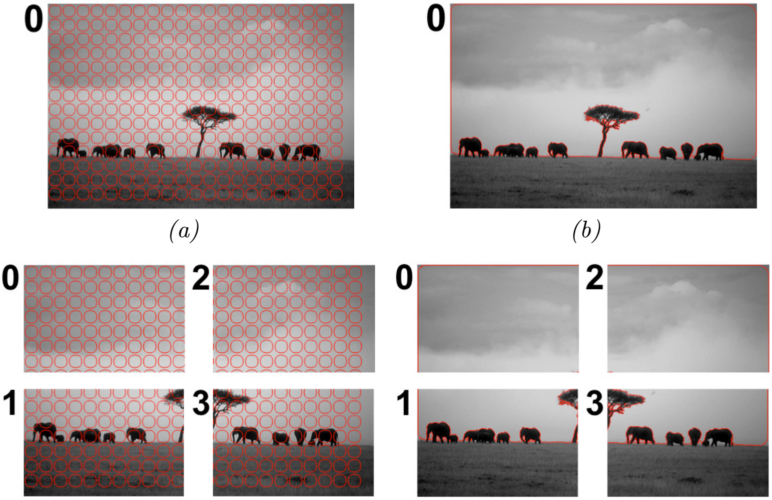


Figure 3.22: Energy evolution of the sequential DRC algorithm of Cardinale et al. (2012) in comparison with the present distributed algorithm processing the piecewise smooth test image from Fig. 3.21 on four and eight processors.

There is also a small difference at energy level in computing the PS energy value by the sequential DRC algorithm using source code of Cardinale et al. (2012) (an image filter in the Insight Toolkit (ITK) image-processing library (Ibanez et al., 2005)) and the present implementation. In our implementation at the image border for spherical patches we consider the outside pixels with zero intensity while they are cut in (ITK) image-processing library. This difference does not change anything in the algorithm as we are only interested in the energy difference and not on its absolute value.

Figure 3.23 illustrates the sequential and distributed segmentations of a natural-scene image using the PS image model. By pixel-to-pixel comparison, the segmentation result on four processors (Fig. 3.23d) is identical to the one computed by a single computer (Fig. 3.23b).



(a) *(b)*

(c) *(d)*

Figure 3.23: Distributed segmentation of a natural-scene image using a piecewise smooth image model. (a) Initialization on a single processor with particles shown in red. (b) Final result on a single processor. (c) Initialization on four processors. (d) Final result on four processors. The two results are identical.

As a first 3D test image, we use the same fluorescence confocal image of zebrafish germ cells that was also used by Cardinale et al. (2012). Figure 3.24 shows the raw image along with the PS segmentation results on one and four processors. By pixel-wise comparison, the results are identical.

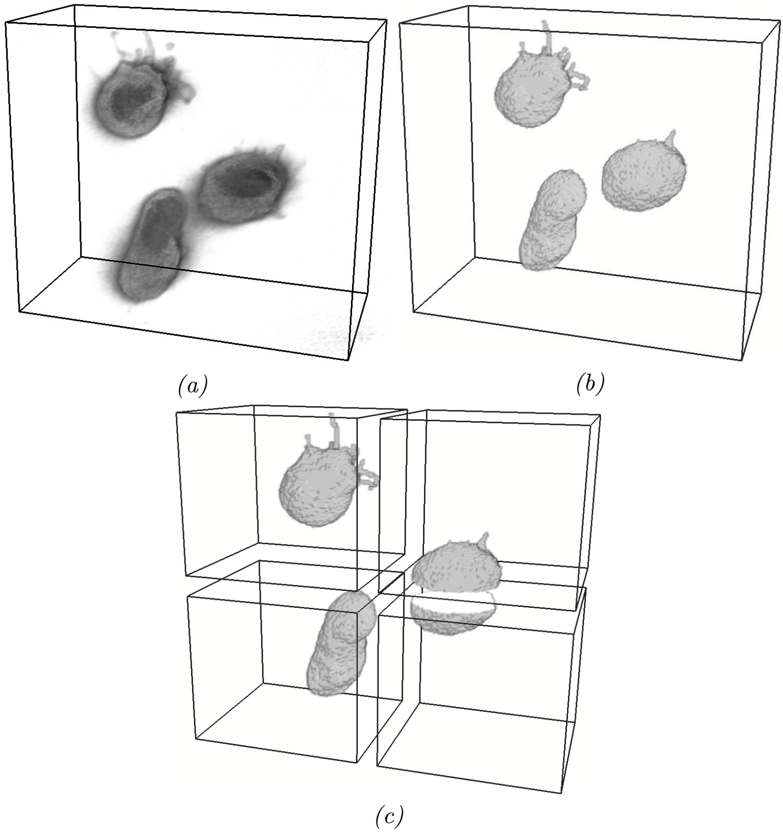


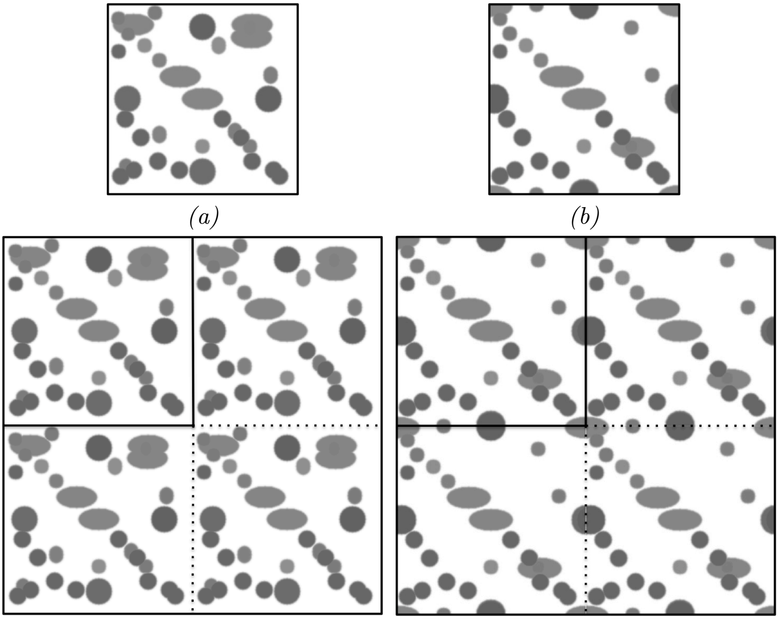
Figure 3.24: Distributed segmentation of zebrafish primordial germ cells using a piecewise smooth image model. (a) Raw 3D confocal fluorescence microscopy image showing three cells with a fluorescent membrane stain (image: M. Goudarzi, University of Münster). (b) Segmentation result on a single processor. (c) Segmentation result on four processors. The two results are identical.

3.3.2 Efficiency of the distributed algorithm

Performance of a distributed parallel algorithm is influenced by many factors, including the structure of the input data, computer memory architecture, communication network, disk space, and background load. While it is impossible to reproduce or control all of these, we present empirical tests in order to assess the overall performance of the present algorithm in terms of parallel scalability and speed. Scalability (parallel efficiency) quantifies how well a distributed algorithm utilizes the computer resources as the number of computers/processors increases. Therefore, we provide results for both weak and strong scaling on synthetic benchmark images. Weak scaling measures how well the algorithm scales to very large images that do not fit into the memory of a single computer. Strong scaling measures how quickly the algorithm can solve a problem of fixed size when it is distributed across an increasing number of computers. We use synthetic images in order to control for variations in the result stemming from image contents. Moreover, synthetic images can easily be scaled to arbitrary size, as required for the weak scaling tests.

The two synthetic images used here are shown in Fig. 3.25. Both are 3D images, and Fig. 3.25 shows maximum-intensity projections. The top row in Fig. 3.25 shows the "unit cells", from which the test images are generated by periodic concatenation as shown in the panels below. In the first image (Fig. 3.25a), all objects are local, i.e., there are no objects that cross sub-image boundaries. The second image (Fig. 3.25b) contains objects that cross sub-image boundaries. Comparing the results of the two allows us to estimate the communication overhead from the parallel graph-handling and region labeling algorithms introduced here. In all cases, the algorithm is initialized with circular regions around each local intensity maximum after blurring the image with a Gaussian filter of $\sigma = 5$ pixel.

In the weak scaling, the workload per processor remains constant, while the overall image size increases proportionally to the number of processors. This way, the workload on 512 processors is an image of $8192 \times 4096 \times 256$ pixels containing 18 944 objects. Periodically repeating the "unit cell" image, rather than scaling it, ensures that the workload on each processor is exactly the same, since every processor locally "sees" the same image.



(a)

(b)

(c)

(d)

Figure 3.25: Maximum-intensity projections of the synthetic 3D test images used to assess the parallel performance and scalability of the distributed algorithm. (a) $256 \times 256 \times 256$ *pixel unit cell of the first test image where no object touches or crosses the boundary. The image contains 37 ellipsoidal objects of different intensities. All objects are non-overlapping in 3D, even though they may appear overlapping in the maximum projection shown here. (b)* $256 \times 256 \times 256$ *pixel unit cell of the second test image with objects touching and crossing the boundary. The image contains 48 ellipsoidal objects of different intensities. The object number is higher than in the first image, because some objects are partial, but the fraction of FG pixels vs. BG pixels is the same as in (a) in order to keep the computational cost (i.e., the number of particles) constant. (c) Synthetic workload image of type 1, generated from 4 unit cells by periodically concatenating them. (d) Synthetic workload image of type 2, generated from 4 unit cells by periodically concatenating them.*

Figure 3.26 shows the resulting parallel efficiency (weak scaling) for the two test images. For comparison, it also shows the parallel efficiency when using the classical master-slave approach to graph processing (see Fig. 3.10). The master-slave approach does not scale, as the parallel efficiency rapidly drops when using more than 32 processors. This results from the communication overhead due to global communication, and from the additional serialization. The present randomized approach scales for both test images.

Segmentation of the second dataset using the present parallel approach on 1, 64, and 512 processors took less than 12, 24, and 29 seconds respectively, corresponding to image sizes of 32 MB, 2 GB, and 16 GB, respectively, in this weak-scaling test. Comparing the results for the first test image, where no objects cross sub-image boundaries, with those for the second test image reveals that about half of the communication overhead is due to boundary particles.

Strong scaling measures how efficiently a parallel algorithm reduces the processing time for an image of a given and fixed size by distributing it across an increasing number of processors. Since the workload per processor decreases as the number of processors increases in a strong scaling, the relative communication overhead steadily grows. Strong scalability is hence always limited by problem size with large problems scaling better. We therefore show tests for two different image sizes: a moderate image size of $512 \times 512 \times 512$ pixels (black circles in Fig. 3.27) and a large image of $2048 \times 2048 \times 2048$ pixels (red squares in Fig. 3.27).

For the first image of size $512 \times 512 \times 512$ pixel, the decrease in efficiency beyond 8 processors is due to communication between the processors, which increases relatively to the smaller and smaller computational load per processor. A 30-fold speedup is achieved for this image size on 64 processors. On 512 processors, every processor only has a sub-image of size $64 \times 64 \times 64$ pixel with ghost layers of width 5 pixel all around. Segmentation of this image on 8, 64, and 512 processors took 16, 4.2, and 1.6 seconds, respectively.

For the larger image of size $2048 \times 2048 \times 2048$ pixel, segmentation on one processor is not possible, since it would require 62 GB of main memory. On 8 and more processors, segmentation becomes feasible and takes 6870

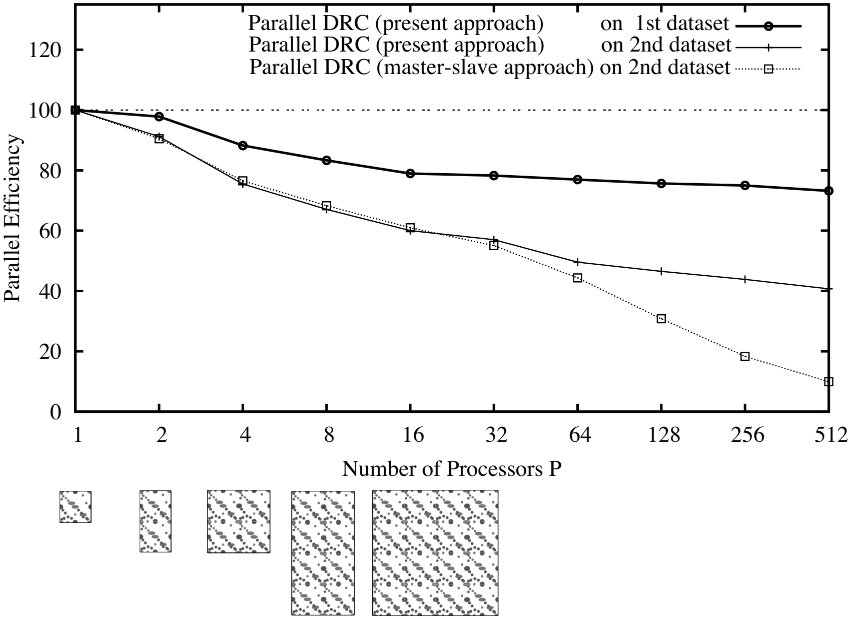


Figure 3.26: Weak scaling parallel efficiency of the present method in comparison with the classical master/slave approach. Time t_1 is the runtime of the algorithm to process a "unit cell" image on one processor, and t_P is the runtime to segment a P-fold larger periodic concatenation image distributed over P processors. The images on 1 to 16 processors are shown below the abscissa for illustration.

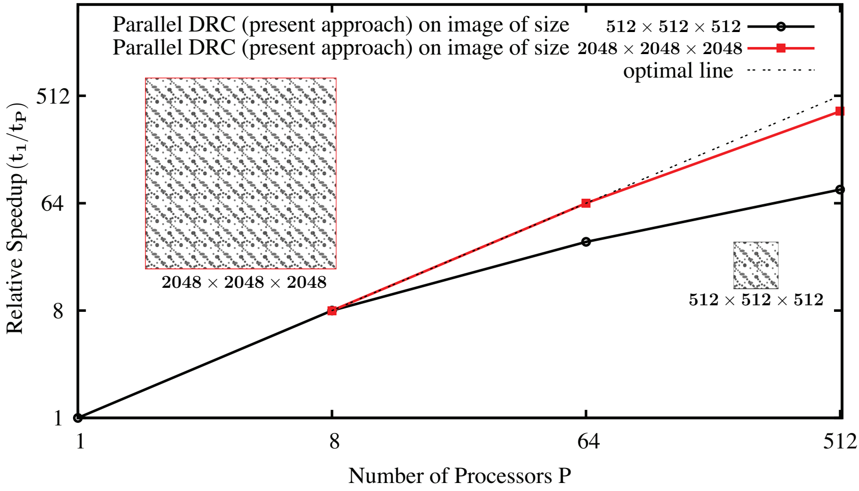


Figure 3.27: Strong scaling speedup versus number of processors P for two different image sizes of $512 \times 512 \times 512$ pixel (black circles) and $2048 \times 2048 \times 2048$ pixel (red squares). The two images are shown in the insets.

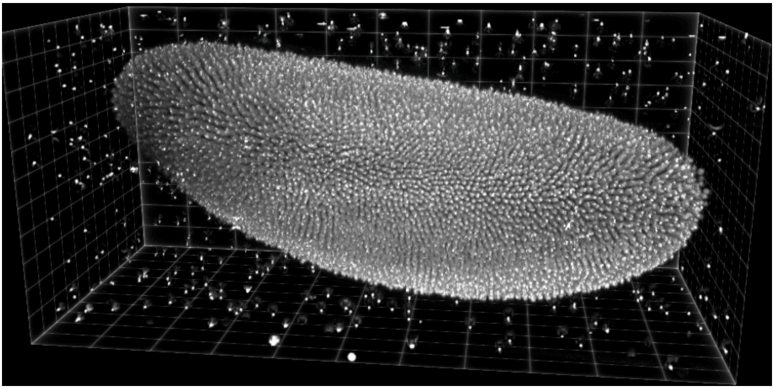
seconds on 8 processors. On 64 and 512 processors, the result is computed in 860 and 145 seconds, respectively. A 48-fold speed is achieved on 512 processors relative to 8 processors, which corresponds to a scalability close to the optimal line.

3.3.3 Application to acquisition-rate segmentation of 3D light-sheet microscopy data

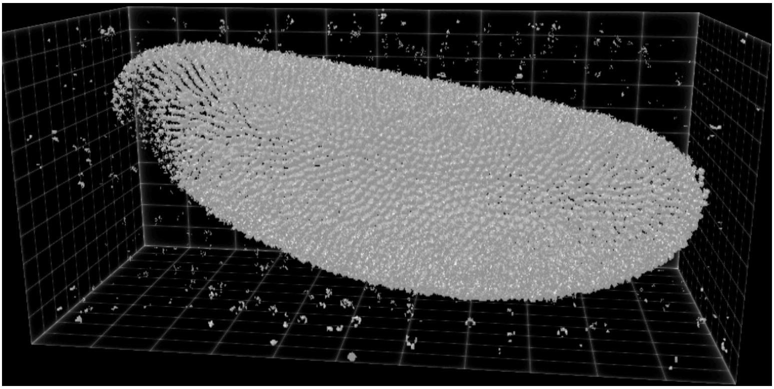
We present a case study applying the present algorithm to segment 3D image data from light-sheet microscopy, demonstrating that acquisition-rate segmentation is possible. We use the image data shown in Figs. 3.28a and 3.30a. Both are 3D light-sheet fluorescence microscopy images.

The first image shows a whole live *Drosophila melanogaster* embryo at cellular blastoderm stage with nuclei labeled by a histone marker. This data was acquired on an OpenSPIM microscope (Pitrone et al., 2013) in the Tomancak lab at MPI-CBG. The size of the original image file is 4.6 GB at 32 bit depth. The image has 1824 × 834 × 809 pixels. During the segmentation, a total of about 64 GB of main memory is required for DRC. Distributed across 128 processors, this is 500 MB per processor, which fits the memory of the individual cores. The segmentation results using the present distributed algorithm with the PC image model on 128 processors is shown in Fig. 3.28b.

Figure 3.28c shows the sub-image from one of the processors, and Fig. 3.28d the corresponding part of the segmentation result. Communication across sub-image boundaries ensures that the processors collectively solve the global problem without storing all of it.

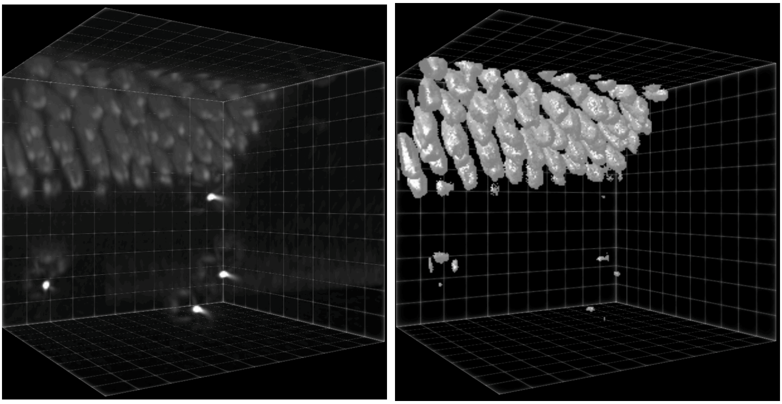


(a)



(b)

Figure 3.28 (continued)



(c) *(d)*

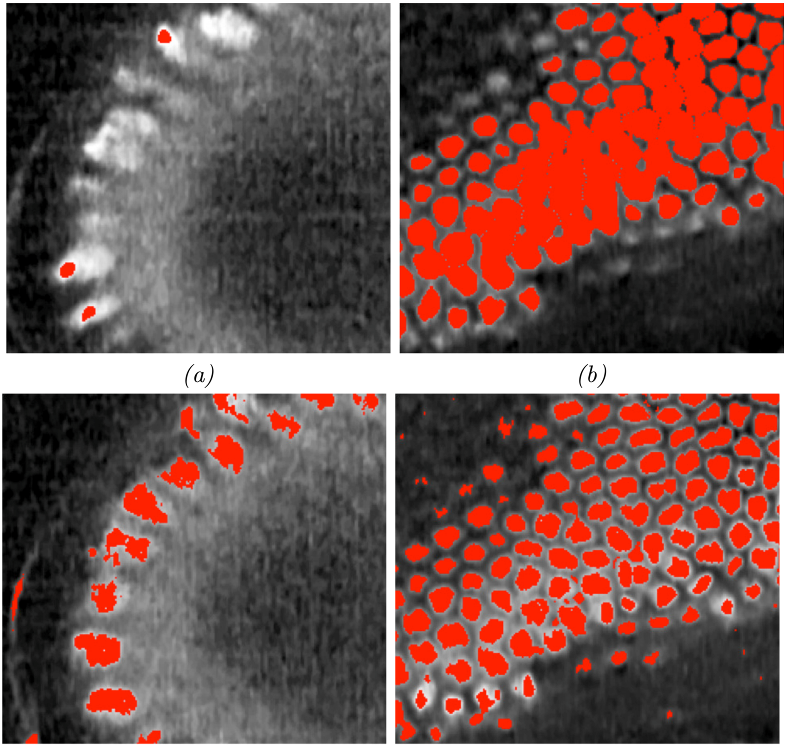
Figure 3.28 (previous page)*: Application of the present implementation to acquisition-rate segmentation of a 3D light-sheet microscopy image using a piecewise constant (PC) image model. All 3D visualizations were done using* ClearVolume *(Royer et al., 2015).*

(a) Raw image showing a Drosophila melanogaster embryo at cellular blastoderm stage with fluorescent histone marker (image: Dr. Pavel Tomancak, MPI-CBG). In addition to the nuclei, there are fluorescent beads embedded around the sample as fiducial markers for multi-view fusion and registration (Preibisch et al., 2010). (b) Segmentation result using the present distributed implementation of DRC with the PC image model distributed across 128 processors. The total time to compute the segmentation was 60 seconds, while the microscope acquired a 3D image every 90 seconds. (c) Example of a sub-image from one of the processors. (d) Corresponding part of the segmentation as computed by that processor.

Segmenting this image distributed across 128 processors took less than 60 seconds, which is shorter than the time of 90 seconds until the microscope acquires the next time point. We hence achieve acquisition-rate image segmentation in this example, using a state-of-the-art model-based segmentation algorithm that produces high-quality results. If necessary, more processors can be used to further reduce processing time, as we have shown the present implementation to scale well up to 512 processors.

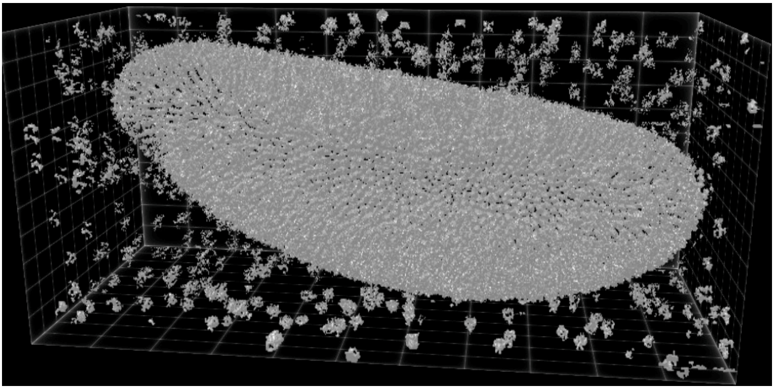
We compare our approach with the TWANG (Stegmaier et al., 2014) pipeline on 14 cores of one compute node (TWANG uses shared-memory multi-threading). TWANG (Stegmaier et al., 2014) required 24 minutes to compute the segmentation using the 14 cores, which does not allow acquisition-rate processing. The comparison is mainly in terms of computational performance, since TWANG was optimized for segmenting spherical objects, whereas the nuclei in our image are rather elongated. The result from DRC hence shows better segmentation quality.

Due to the inhomogeneous fluorescence intensity across the sample, the PC segmentation shown in Fig. 3.28b misses some of the nuclei at the left tip of the embryo. This can be improved using the PS image model instead, which allows for intensity gradients within regions, in particular within the background region. This is shown in Fig. 3.29. Figure 3.29a shows a low-intensity region where the PC model misses some nuclei. Figure 3.29b shows a high-intensity regions where the PC model fuses several nuclei together. The corresponding results when using the PS image model are shown in the panels below, in Figs. 3.29c and 3.29d. The whole-image result when using the PS image model is shown in Fig. 3.29e. The PS model improves the segmentation since it adjusts to local intensity variations in the objects and the background, which is also why it captures more of the fiducial beads around the embryo. This demonstrates the flexibility of DRC to accommodate for different image models, enabling application-specific segmentations that include prior knowledge about the image. The segmentation quality can further be improved by including shape priors (Zhang and Lu, 2004; Veltkamp and Hagedoorn, 2001), as has been demonstrated for DRC (Cardinale, 2013), or by using Sobolev gradients for which DRC is uniquely suited (Sbalzarini et al., 2014).



(a) *(b)* *(c)* *(d)*

Figure 3.29 (continued)

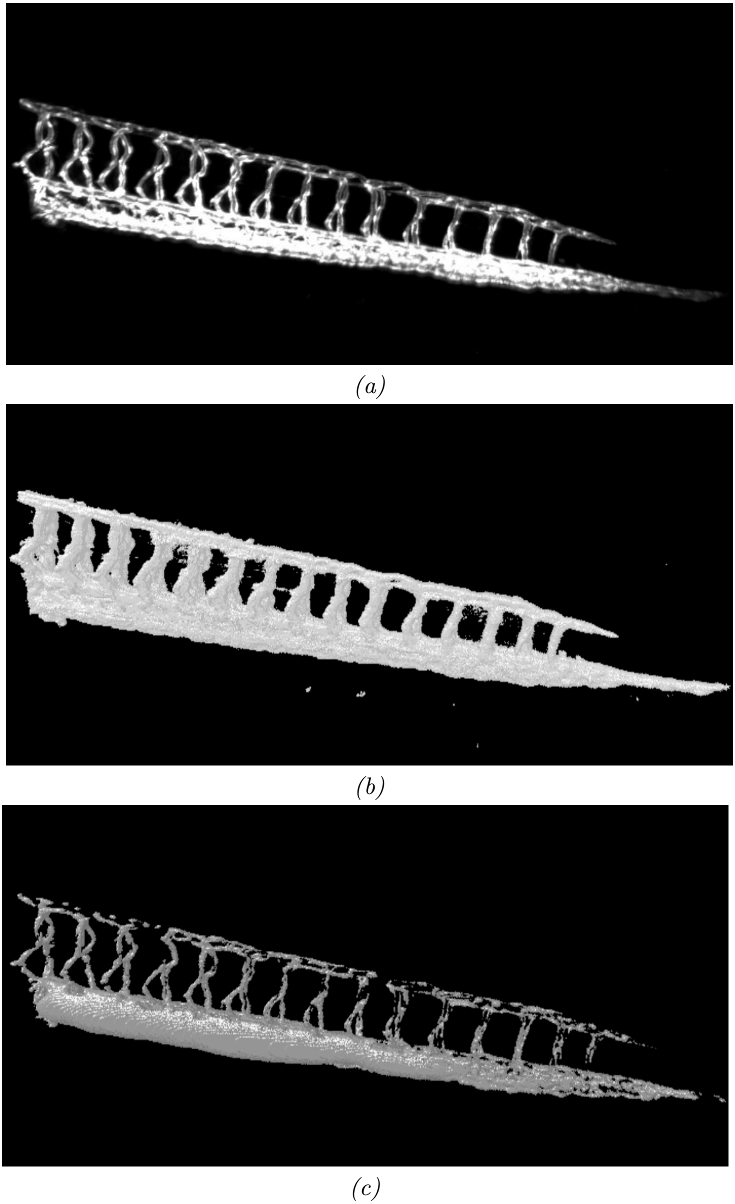


(e)

Figure 3.29 (previous page)*: Comparison of the piecewise constant (PC) and piecewise smooth (PS) image models. All visualizations were done using* ClearVolume *(Royer et al., 2015). (a) Segmentation (red) overlay in a low-intensity region using the PC model. (b) Segmentation (red) overlay in a high-intensity region using the PC model. (c) Segmentation (red) overlay in the same low-intensity region using the PS model. (d) Segmentation (red) overlay in the same high-intensity region using the PS model. (e) Complete result using the PS model on 128 processors. The total processing time was 250 seconds for this case.*

Using the PS image model, however, is computationally more involved than using the PC model. The segmentation shown in Fig. 3.29e required 250 seconds to be computed on 128 processors. Acquisition-rate processing using the PS model hence requires about 512 processors.

The second image shows the tail of a live zebrafish embryo 3.5 days post fertilization with the vasculature fluorescently labeled by expressing GFP in endothelial cells (Tg(flk1:EGFP)s843). This image was acquired by the Huisken lab at MPI-CBG using a state-of-the-art light-sheet microscope (Huisken et al., 2004). The geometric structure of a vascular network is very different from blob-like nuclei, illustrating the flexibility of DRC to segment arbitrary shapes. This image is intractable for specialized blob-segmentation pipelines like TWANG (Stegmaier et al., 2014). Figure 3.30a shows the raw data. The image has 1626 × 988 × 219 pixels. Figure 3.30b shows the segmentation result using Li's minimum cross-entropy thresholding (Li and Lee, 1993), as implemented in the software package *Fiji* (Schindelin et al., 2012). We use this thresholding as an initialization for our method. Figure 3.30c is the segmentation result using the PS image model with a Gaussian noise model. Distributed processing on 32 processors took 248 seconds. In this segmentation, some vessels appear non-contiguous and the caudal vessels (caudal artery and caudal vein) are not properly resolved. This changes when replacing the Gaussian noise model with a Poisson noise model (Paul et al., 2013), as shown in Fig. 3.30d. Using the correct noise model clearly improves the result, providing further illustration that flexible frameworks like DRC are important. The result in Fig. 3.30d was obtained on 32 processors in less than 200 seconds.

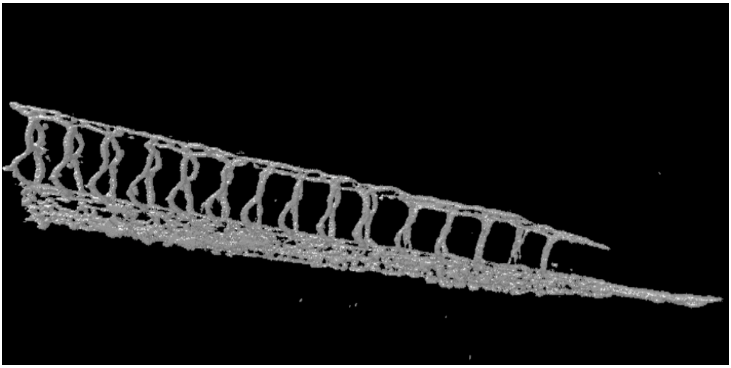


(a)

(b)

(c)

Figure 3.30 (continued)



(d)

Figure 3.30 (previous page)*: Application of the present method to segmenting zebrafish vasculature in a light-sheet microscopy image. All visualizations were done using* ClearVolume *(Royer et al., 2015). (a) Raw image showing the tail part of the vasculature in a developing zebrafish embryo 3.5 days post fertilization (image: Stephan Daetwyler, Huisken lab, MPI-CBG). (b) Initialization using Li thresholding (Li and Lee, 1993). (c) Segmentation result using the PS image model with Gaussian noise model on 32 processors. The total processing time was 248 seconds. (d) Segmentation result using the PS image model with Poisson noise model (Paul et al., 2013) on 32 processors. The total processing time was less than 200 seconds.*

3.4 SUMMARY

We have presented a distributed-memory parallel implementation of the Discrete Region Competition (DRC) algorithm (Cardinale et al., 2012) for image segmentation. Efficient parallelization was made possible by a novel parallel independent sub-graph algorithm, as well as optimizations to the parallel connected-component labeling algorithm. The final algorithm was implemented using the PPM library (Sbalzarini et al., 2006; Awile et al., 2010, 2013) as an efficient middleware for parallel particle-mesh methods. The parallel implementation includes both piecewise constant (PC) and piecewise smooth (PS) image models.

The distributed-memory scalability of the presented approach effectively overcomes the memory and runtime limitations of a single computer. None of the computers or processors over which a task is distributed needs to store the entire image. This allows segmenting very large images. The largest synthetic image considered here had $1.7 \cdot 10^{10}$ pixels, corresponding to 32 GB of uncompressed memory. A real-world light-sheet microscopy image of 1824 × 834 × 809 pixels (amounting to 4.6 GB of uncompressed memory) was segmented in under 60 seconds when distributed across 128 processors. This was less than the 90 seconds until the microscope acquired the next time point, hence providing online, acquisition-rate image analysis. This is a prerequisite for smart microscopes (Scherf and Huisken, 2015) and also enables interactive experiments.

We have demonstrated the parallel efficiency and scalability of the present implementation using synthetic images that can be scaled to arbitrary size. We have further reproduced the benchmark cases from the original DRC paper (Cardinale et al., 2012) and have shown that the parallel implementation produces the same or very close results as the original sequential reference implementation. Small differences may occur, but are limited to isolated oscillatory pixels, which are due to local oscillation detection. This local detection is preferable because it avoids global communication and improves parallel scalability with respect to the traditional master/slave approach.

Although our performance figures are encouraging, there is room for further improvements. One idea could be to compress the particle and image data before communication. This would effectively reduce the communication overhead and improve scalability. Furthermore, spatially adaptive domain decompositions and dynamic load balancing could be used to reduce load imbalance. Depending on the image contents, not all processors may have an equal amount of particles. This causes asynchronous waiting times that may limit scalability. Due to the checkerboard decomposition used in the graph handling, however, the current implementation is limited to Cartesian domain decompositions, while spatially adaptive trees might be better. Lastly, the local evaluation of energy differences for all possible particle moves can be accelerated by taking advantage of multi-threading and graphics processing units (GPUs). This is possible for DRC, as has already been shown (Ebrahim, 2011), suggesting that processing could be further accelerated by a factor of 10 to 30, depending on the image model.

This leaves ample opportunities for further reducing processing times as required by the microscopy application. Already the present implementation, however, illustrates the algorithmic concept, which is based on randomized graph decomposition and hybrid particle-mesh methods. This enables acquisition-rate segmentation of 3D fluorescence microscopy images using different image models, opening the door for smart microscopes and interactive, feedback-controlled experiments.

CHAPTER

FOUR

PARALLEL DISTRIBUTED-MEMORY DISCRETE REGION SAMPLING ALGORITHM

4.1 INTRODUCTION

In the previous chapter, we presented a distributed-memory parallel implementation of the DRC algorithm for image segmentation. The presented parallel approach overcomes the memory and runtime limitations of a single computer. However, no information is provided about the sensitivity or robustness of the segmentation.

Accuracy and robustness of segmentation affect the quantitative results in an image analysis pipeline. In a pipeline, uncertainty quantification of the segmentation is crucial. Knowing the uncertainty in the results minimizes user interactions by highlighting critical locations in the solution and their quality. Uncertainty quantification of segmentation can provide a measure to guide the user to areas where the uncertainty is highest.

DRS provides such a measure of segmentation uncertainty. It does this by MCMC sampling from the posterior around the DRC segmentation result. The proposed transitions move the Markov chain from state X_t to X_{t+1}, by small alterations. Knowing that the stationary distribution of the Markov Chain is the segmentation posterior, we can construct the transition kernel and calculate the probability of applying each move. The length of the burn-in phase, or the number of iterations required for convergence, is an unsolved problem and beyond the scope of this thesis. Empirically, it takes thousands of iterations for the chain to converge, and then it needs to explore enough of the state space to detect and assign probabilities to alternative modes. How many samples are required to detect alternative modes is uncertain.

Here, we address the runtime and memory issues of DRS by parallelizing it across multiple computers. Parallelization at a first glance is an impossible task, since each state in the chain depends on the previous state. Thus, subsequent moves can not be performed independently. Nevertheless, several attempts at parallelization have been taken previously.

4.1.1 Previous approaches

There are several approaches to parallelization of MCMC for image analysis. We review the traditional methods of Rosenthal (2000) and Geyer (1991) and give an overview of the more recent method of Byrd (2010). We describe why these methods are not suitable here. Then, among many available MCMC algorithms for particle simulations, we review the recent works of Heffelfinger and Lewitt (1996); Anderson et al. (2013, 2016), where the parallelization strategy relies on a domain-decomposition scheme. We describe why these algorithms are not suitable for parallel MCMC sampling in image analysis, but, they inspire us to develop our new parallel algorithm.

Parallel processing strategies for MCMC in image analysis can be designed in various ways. The traditional way is the typical Monte Carlo parallelization (Rosenthal, 2000). In this way, the simulation starts using multiple chains on multiple computers, each with a separate initial state and random number generator. In this method, even though multiple chains run on multiple computers independently, the average burn-in time for each chain does not change. Therefore, it does not solve the runtime nor memory issues on the extremely large state spaces of images.

Another approach to reducing the runtime of MCMC applications therefore is to improve the convergence rate. Geyer (1991) proposed a technique known as Metropolis-Coupled MCMC that promotes mixing by using multiple MCMC chains with different stationary distributions. From time to time, chains exchange their states according to the Metropolis-Hastings criterion. The parallel implementation runs each chain on a different processor. This approach reduces the number of iterations required for the chains to converge, but it does not solve the runtime nor memory issues on large state spaces.

Byrd (2010) introduced a periodic parallelization approach. This method at first performs all the sequential MCMC moves that are not possible to do in parallel. This includes everything that alters the configuration or impacts calculations across the entire image (global moves). Then, it randomly partitions the image and simultaneously continues in each partition, performing the moves that are possible to do in parallel. These moves only affect a small area and make no changes to the global properties (local moves). After recombining the changes in all partitions into the whole image, the cycle repeats starting again with global MCMC moves on the whole image. The sequential (global) moves make large-scale alterations to the image, while the parallelized (local) moves fine tune specific features. This cycle repeats until the partitioning into global and local moves is statistically insignificant. The random partitioning prevents boundary anomalies. Even though this approach scales and produces reliable results, it is limited by the serialization of the global moves on the master node. Those extensions of this approach either require preprocessing the whole image, which is limited to images with independent features, or requires post-processing on the partitions which can create artifacts from the partitioning (Byrd, 2010).

Parallelizing MCMC simulations using particles is widely considered. A number of works for particle simulations in systems with short ranged interactions used checkerboard techniques (Heffelfinger and Lewitt, 1996; Anderson et al., 2013, 2016).

Heffelfinger and Lewitt (1996) presented a spatial decomposition approach, in which the sub-images are further divided into 2^d cells (where d is the dimension of the space). Figure 4.1 illustrates this for four sub-images divided into sixteen cells. By choosing the cell size sufficiently large, each processor can perform MCMC sampling independent of its neighboring processors. For example, all `a`-cells in different sub-images (and thus on different processors) are separated by at least one cell. Therefore, sampling in the `a`-cells of neighboring sub-images can be done independently if the cell edge length is larger than the radius of the energy kernel. Figure 4.2 illustrates an example of sampling two particles simultaneously in the `a`-cells of two neighboring sub-images.

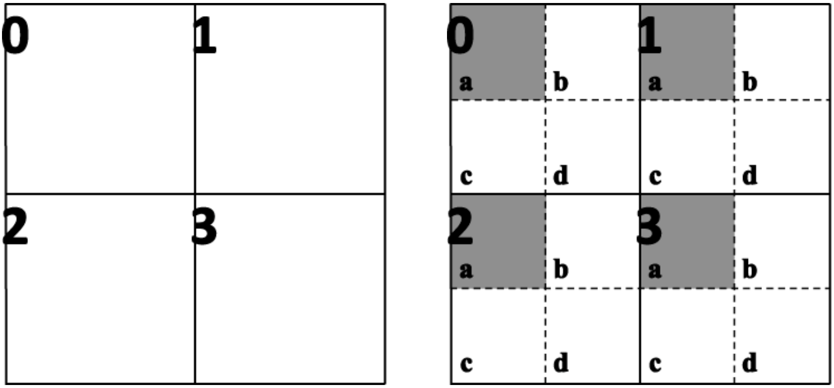


Figure 4.1: An image with four sub-images (on four processors) divided into sixteen cells.

Each processor works first in the `a`-cell, in the corresponding sub-image, sampling a test move (i.e., a test-particle). The processors then successively visit the other cells and continue sampling in the `b,c,` and `d`-cells, respectively. Each processor then sends its test-particles to its neighbors and receives test-particles from them. Using this information, sampling completes by accepting or rejecting the proposed test-particles. The boundary information between sub-images is communicated between the respective processors with ghost layers.

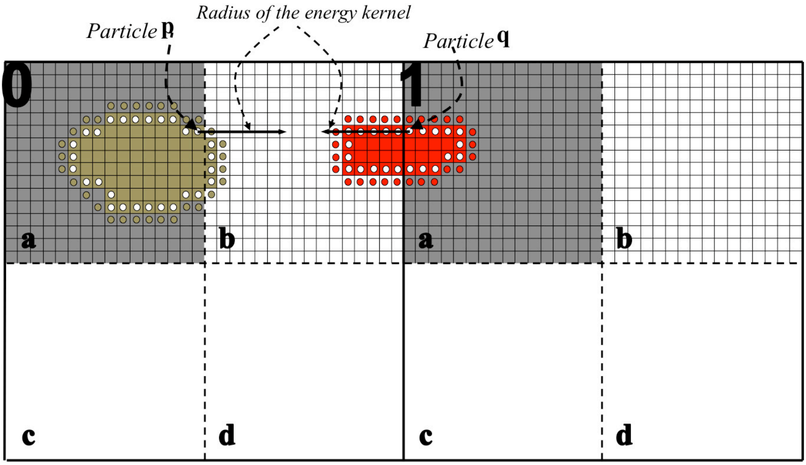


Figure 4.2: An example of sampling two particles simultaneously in two sub-images (different processors). Both processors (with IDs 0 *and* 1*) are applying the sampled particles moves (***p** *and* **q***) independently on their* 𝔞*-cells.*

Due to the extra memory required for the test particles, however, this approach is not suited for the large state space of images. It also causes waiting times for the completion of the additional send and receive operations between neighboring processors. More importantly, it introduces a bias in the sampling process.

Anderson et al. (2013) addressed the bias problem, caused by sampling cells in order, and presented a parallel MCMC for particle simulations on GPUs. They used a checkerboard decomposition with a cell-list data structure. They shuffled the checkerboard sets and assured a random permutation. They showed medium-scale simulations on GPUs and assessed the scalability of the approach. Anderson et al. (2016) generalized the domain decomposition parallelization on many CPUs and many GPUs using MPI and to simulations with large particle size disparity. In this parallelization, each processor domain is divided into an active and an inactive area, where the right and bottom edges of each domain (in 2D) are inactive areas. After N trial moves on the active area, and accepting or rejecting the moves, all the particles are shifted with a random displacement vector. This is followed by migrating the particles to new processors if required, and updating the ghosts.

All of these works do not directly apply to MCMC sampling in the large state spaces of images. They either do not solve the runtime and memory issues or, in the case of Anderson et al. (2016), they are not suited for images where the objects are fixed.

Our goal here is to find a representative collection of segmentations and to estimate the uncertainty of the output due to the variability of the segmentations in the large state space of the image. We draw inspiration from the previous works of Heffelfinger and Lewitt (1996); Anderson et al. (2013, 2016) and introduce a novel distributed-memory MCMC approach. We parallelize the DRS algorithm (Cardinale, 2013) for uncertainty quantification of segmentations in a distributed environment by applying a domain-decomposition approach to the image.

Our solution is appropriate for image models with limited computational support, such as piecewise constant and smooth energies, where energy computation does not require the global information over the image. We describe the design and implementation of the parallel approach. Our approach is specifically designed to support parallel execution on heterogeneous clusters and distributed-memory machines.

4.2 Algorithm

We again design the algorithm using a domain-decomposition approach in order to scale to large numbers of processors. Thus, we identify the computation that can be executed simultaneously in each sub-image. Efficient simultaneous computation requires the development of a parallel algorithm that minimizes inter-processor communication and synchronization. We use a checkerboard decomposition to organize independent sets that can be executed simultaneously without conflicts between neighboring processors.

4.2.1 Domain decomposition

The domain decomposition for uncertainty quantification takes advantage of finite computational support of the energy. Processors can thus work simultaneously, as long as the regions in which they operate are separated by a distance larger than the number of pixels required to compute energy differences, i.e., by the radius of the energy kernel.

The input image is decomposed into disjoint sub-images that are distributed to different computers (see Section. 3.2). Domain decomposition and data distribution are done transparently by the PPM library (Sbalzarini et al., 2006; Awile et al., 2010, 2013). Reading the input image from a file is also done in a distributed way, where each computer only reads certain image planes. The PPM library then automatically redistributes the data so as to achieve a good and balanced decomposition. Each computer only stores its local sub-image, and no computer needs to be able to store the entire image data.

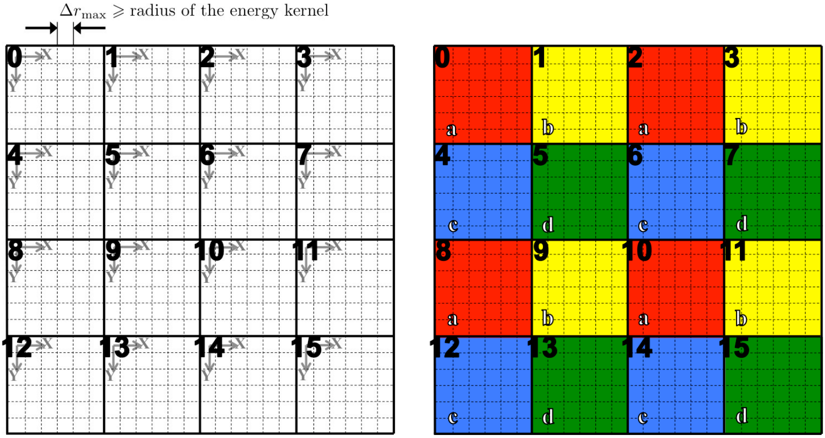
The algorithm is then initialized locally on each computer, using only the local sub-image. The initial label image L_0 is the result of any previous segmentation, like results from DRC, manual, or semi-manual segmentations, or any other initialization approach. Good initializations start the chain close to the posterior equilibrium distribution.

A significant difference to the DRC algorithm is that we need to fix the number of regions for sampling. This is because we want to sample a segmentation space of known dimension.

4.2.2 Checkerboard decomposition

A two-dimensional example of a Cartesian domain decomposition of an image into sixteen sub-images is depicted in Fig. 4.3. In each sub-image, we create a cell list data structure. Cell lists work by dividing each sub-image into equisized cells with edge length $\Delta r_{\max}$, greater than or equal to the radius of the energy kernel. This ensures that cells to be updated independently (Heffelfinger and Lewitt, 1996).

Using this structure, we describe our algorithm for parallel updates and asynchronous communication between processors.



(a) *(b)*

Figure 4.3: An image distributed into sixteen sub-images on sixteen processors. (a) Each sub-image is partitioned into cells, where the edge length $\Delta r_{\max}$ *is bigger than the radius of the energy kernel. (b) A greedy graph coloring (Coleman and Moré, 1983) algorithm defines the independent sets (four colors* $a, b, c,$ *and* d *in two-dimensional image) across processors. The white letter on each sub-image is the color label.*

We first assign the sub-images into sets that can be updated independently. To determine independent sets of sub-images, we use the fast greedy graph coloring heuristic of Coleman and Moré (1983). Graph coloring assigns labels "colors" to vertices of a graph such that no two adjacent vertices have the same color (Diestel, 2012). Here, every sub-image (processor) is a vertex of a graph and adjacency with other sub-images (neighbors) introduces edges between the vertices. Figure 4.3b shows the results of graph coloring so that no two adjacent sub-images have the same color (four and eight colors in two-dimensional and three-dimensional images, respectively). This determines the independent sets of processors, where each color indicates a label $(a, b, \cdots)$. This step is done once at the start of the program and is valid for arbitrary domain decompositions, also beyond Cartesian ones.

Second, in each sub-image, we identify a cell of the local cell list by its coordinate ((x, y) and (x, y, z) in two and three-dimensional sub-images respectively). The coordinates of each cell determine its checkerboard set (Anderson et al., 2013) as,

$$\mathcal{C}_{c2D} = \begin{cases} a & \text{if } (x \text{ is even}) & \text{and} & (y \text{ is even}), \\ b & \text{if } (x \text{ is odd}) & \text{and} & (y \text{ is even}), \\ c & \text{if } (x \text{ is even}) & \text{and} & (y \text{ is odd}), \\ d & \text{if } (x \text{ is odd}) & \text{and} & (y \text{ is odd}), \end{cases}$$

$$\mathcal{C}_{c3D} = \begin{cases} a & \text{if } (x \text{ is even}) & \text{and} & (y \text{ is even}) & \text{and} & (z \text{ is even}), \\ b & \text{if } (x \text{ is odd}) & \text{and} & (y \text{ is even}) & \text{and} & (z \text{ is even}), \\ c & \text{if } (x \text{ is even}) & \text{and} & (y \text{ is odd}) & \text{and} & (z \text{ is even}), \\ d & \text{if } (x \text{ is odd}) & \text{and} & (y \text{ is odd}) & \text{and} & (z \text{ is even}), \\ e & \text{if } (x \text{ is even}) & \text{and} & (y \text{ is even}) & \text{and} & (z \text{ is odd}), \\ f & \text{if } (x \text{ is odd}) & \text{and} & (y \text{ is even}) & \text{and} & (z \text{ is odd}), \\ g & \text{if } (x \text{ is even}) & \text{and} & (y \text{ is odd}) & \text{and} & (z \text{ is odd}), \\ h & \text{if } (x \text{ is odd}) & \text{and} & (y \text{ is odd}) & \text{and} & (z \text{ is odd}), \end{cases}$$

where, a,b,$\cdots$ also indicate the labels of the checkerboard sets. So $\mathcal{C}_c = \{a, b, c, d\}$ for two-dimensional images.

In contrast to Cardinale (2013), in our strategy particle positions (and not region labels) determine the order of updates. The order will thus change as region contours change. Consequently, care is required to ensure unbiased results.

We draw many samples in parallel (Heffelfinger and Lewitt, 1996). Therefore, we also need to generate independent parallel random number streams.

4.2.3 Parallel Pseudo random number generation

For unbiased sampling, we use a Mersenne Twister random number generator (MTRNG) (Matsumoto and Nishimura, 1998). To assure a unique seed on every processor, we mix the processor ID and the user seed with the mixing procedure of a hash-based random number generator, Saru (Afshar et al., 2013), as:

```
seed2+=seed1<<16;
seed1+=seed2<<11;
seed2+=((signed int)seed1)>>7;
seed1^=((signed int)seed2)>>3;
seed2*=0xA5366B4D;
seed2^=seed2>>10;
seed2^=((signed int)seed2)>>19;
seed1+=seed2^0x6d2d4e11;
seed1 = 0x79dedea3*(seed1^(((signed int)seed1)>>14));
seed2 = (seed1 + seed2) ^ (((signed int)seed1)>>8);
seed1 = seed1 + (seed2*(seed2^0xdddf97f5));
seed2 = 0xABCB96F7 + (seed2>>1);
seed1 = 0x4beb5d59*seed1 + 0x2600e1f7;
seed2 = seed2+0x8009d14b + ((((signed int)seed2)>>31)&0xda879add);
MTseed=(seed1 ^ (seed1>>26))+seed2;
MTseed=(MTseed^(MTseed>>20))*0x6957f5a7;
```

where `seed1` and `seed2` are set to the processor ID and user seed, respectively. `<<` and `>>` are left and right bit-shift operations, respectively, `^` is a bitwise exclusive-OR operation and `&` is a bitwise AND operation.

Seeding a random number generator is a highly nontrivial task, since one has to eliminate all structures of the input seeds (Afshar et al., 2013). The above initial mixing does not create any correlation and `MTseed` is used to seed MTRNG (the test harness ensures that the behavior of this seeding is indeed chaotic). This seeding assures uncorrelated RNG streams on different processor ID. We then use MTRNG to generate as many random numbers as needed for DRS sampling.

To avoid bias in our sampling, we shuffle the independent cell sets. For shuffling, every processor must use the same permutation. This is necessary for consistent communication with neighboring sub-images. To do this, we use the Saru random number Generator (SRNG) (Afshar et al., 2013). At each iteration, we use two input seeds (iteration number and user seed) to initialize an independent RNG stream. Then, we use the seeded RNG to shuffle the order of the cell sets with the Fisher-Yates shuffle algorithm (Durstenfeld, 1964). This shuffling guarantees a random permutation (Anderson et al., 2013), and the permutation results are the same on all processors. Algorithm 5 implements the Fisher-Yates shuffle.

Algorithm 5: Fisher-Yates shuffle

```
// To shuffle an array $\mathcal{A}$ of n elements
for $i \leftarrow n-1$ to 1 do
    $U \leftarrow \boldsymbol{RNG}.uniform(0,1)$;
    $j \leftarrow (int)(i \times U)$;
    Exchange: $a[j]$ and $a[i]$
```

We start sampling by iterating over cells and applying DRS independently to the particles in each cell.

4.2.4 Iteration structure

In each sub-image, we divide the cells into boundary cells and interior cells. This is illustrated in Fig. 4.4.

We use an array $\mathcal{C}_b$ to store all the boundary cell indices and an array $\mathcal{C}_i$ to store all interior cell indices. Interior cell indices are sorted based on their color set label. The range of each set is stored in an array $\mathcal{C}_n$, where $\mathcal{C}_n$ is an integer array (of length five in 2D and nine in 3D), and every entry in the $\mathcal{C}_n$ array specifies the displacement (relative to $\mathcal{C}_i$). For example, suppose that for a two-dimensional image, we have 460 interior cells in one sub-image. Then, $\mathcal{C}_n = \{0, 110, 210, 310, 460\}$ means that the first 110 cell indices, starting from index $0+1$ to 110 in $\mathcal{C}_i$ are indices of `a`-cells and the next 100 cell indices in $\mathcal{C}_i$, starting from index $110+1$ to 210 are indices of `b`-cells, and so on.

Sampling in interior cells can be done independently on each processor. Sampling in the boundary cells, however, requires information from neighboring sub-images to prevent anomalies at the sub-image borders. Concurrent sampling on the border cells of two neighboring sub-images would break the independent sets.

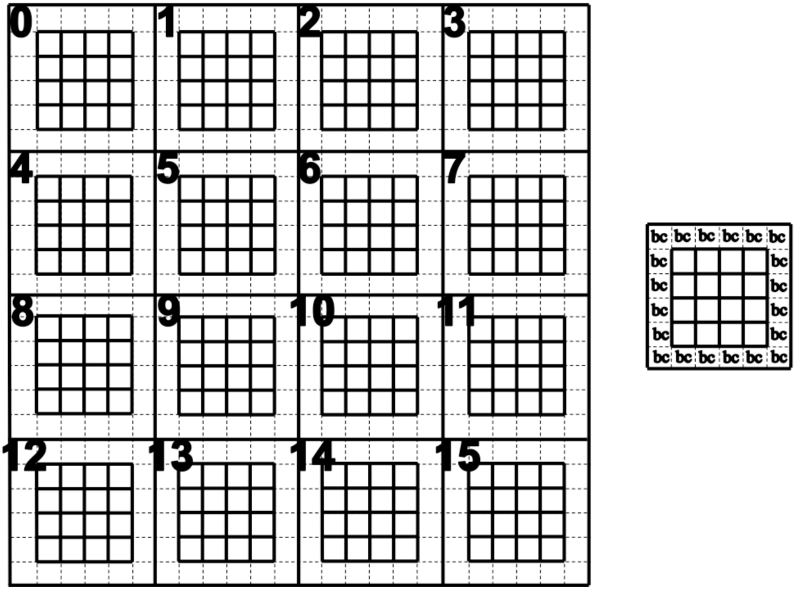
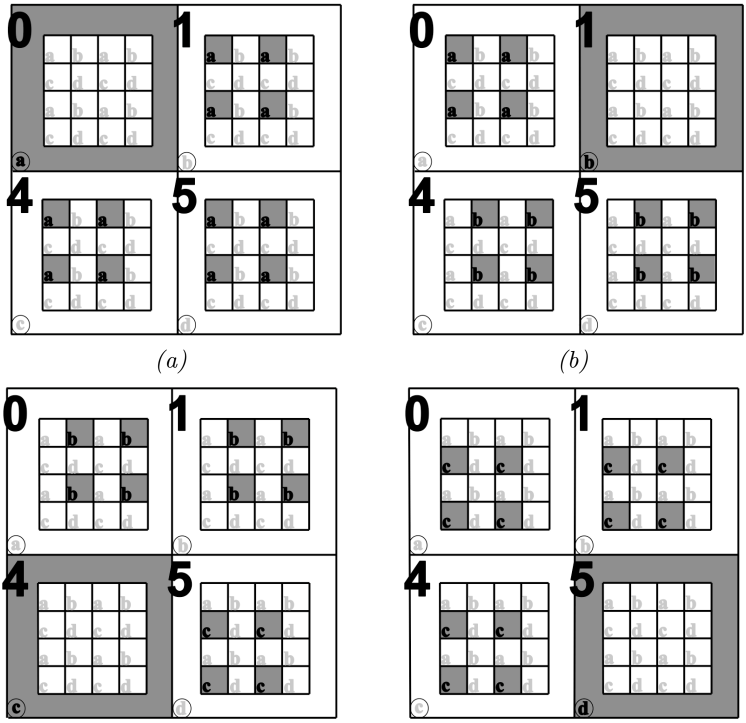


Figure 4.4: Illustration of boundary cells and interior cells in each sub-image. Boundary cells are at the border of each sub-image, indicated by dashed lines. For an example sub-image, boundary cells are marked with ***bc*** *in the right panel.*

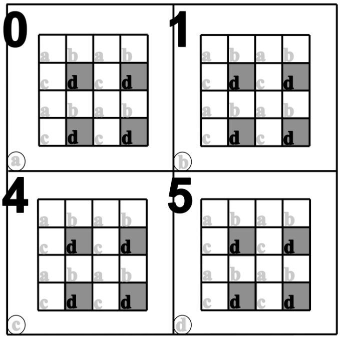
To ensure statistically correct sampling, we design the iterator with sub-sweeps. Every iteration consists of five sub-sweeps over cells in two dimensions (nine in three dimensions). One iteration and its five sub-sweeps are illustrated in Fig. 4.5 for four adjacent sub-images (on four processors) from Fig. 4.4.



(a) *(b)*

(c) *(d)*

Figure 4.5 (continued)



(e)

Figure 4.5 (previous page)*: Illustration of the five sub-sweeps over cells comprise one iteration of our algorithm. Active (selected) cells are separated by one row or by one column of inactive cells. The color of each sub-image is shown in a circle at the corner of the sub-image. Here the independent set* $\mathcal{C}_c = \{a, b, c, d\}$*, and processors ID 0,1,4, and 5 have colors* $a, b, c,$ *and* d *respectively. (a) An active processor ID 0 samples in the boundary cells and processors ID 1,4, and 5 sample in active interior cells. (b) The second sub-sweep, where processor ID 1 with color b is active and samples in the boundary cells, while processors ID 0,4, and 5 sample in active interior cells. (c) The third sub-sweep, where processor ID 4 with color c is active and samples in the boundary cells, while processors ID 0,1, and 5 sample in active interior cells. (d) The fourth sub-sweep, where processor ID 5 with color d is active and samples in the boundary cells, while processors ID 0,1, and 4 sample in active interior cells. Every sub-sweep over the boundary cells is followed by a ghost-layer update communication (see Algorithm 6). (e) Illustration of the last sub-sweep, where all processors sample in active interior cells.*

While processors of the same color sample in their boundary cells, their neighboring processors of different colors sample in their interior cells. Using this strategy, we ensure independent samples.

Algorithm 6 outlines the structure of our parallel strategy at each iteration with the corresponding sub-sweeps.

Algorithm 6: Parallel MCMC Iteration

```
Initialize: $RNG \leftarrow$ Saru(iteration number, user seed)
Shuffle: $\mathcal{C}_c$ using Fisher-Yates shuffle
Shuffle: $\mathcal{C}_b$ using Fisher-Yates shuffle
Require: $N_{\mathrm{p\,max}}$ is the maximum number of particles in any cell
$k \leftarrow 0$;
for $i \leftarrow 1$ to 4 (or 8 in 3D) do
    if processor color is $\mathcal{C}_c(i)$ then
        subsweep($\mathcal{C}_b$);
        Send: non-blocking send of updated label information at
                the boundary to the neighbors
        Wait: for non-blocking send to complete
    else
        Receive: non-blocking receive of ghost information from
                neighbor processors with color $\mathcal{C}_c(i)$

        $k \leftarrow k + 1$;
        subsweep($\mathcal{C}_i$ ($\mathcal{C}_n(k) + 1$ to $\mathcal{C}_n(k+1)$));

        Wait: for non-blocking receive to complete
        Update: the border cells from changes in the updated
                ghosts
$k \leftarrow k + 1$;
subsweep($\mathcal{C}_i$ ($\mathcal{C}_n(k) + 1$ to $\mathcal{C}_n(k+1)$));
```

Algorithm 7: Sub-sweep in cells

```
0 Procedure: subsweep($\mathcal{C}_{in}$)

1 for $c \in \mathcal{C}_{in}$ do
2     $N_p \leftarrow$ len($c.particles$) // number of particles in cell $c$
3     $U \leftarrow$ RNG.$uniform(0, 1)$;
4     if $U < \frac{N_p}{N_{p\max}}$ then
5         DRS sampling from $c.particles[\mathcal{P} \cup \mathcal{P}_f]$
```

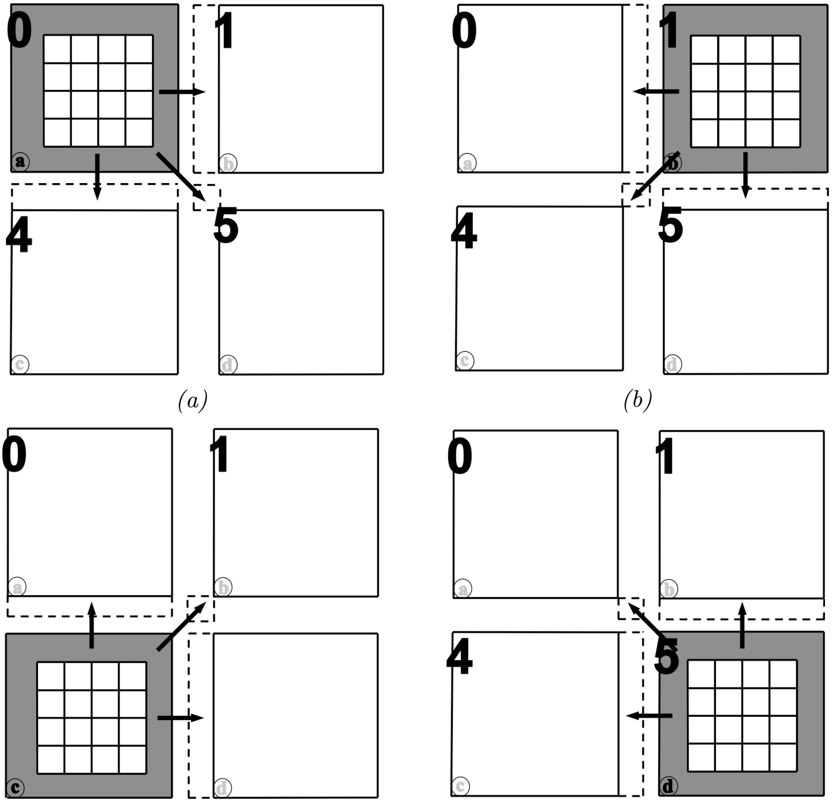
Algorithm 6, line 2 shuffles the array of border cells ($\mathcal{C}_b$). In each iteration, during a sub-sweep i, ($i \leftarrow 1$ **to** 4 (or 8 in 3D)), boundary cells of processors with the same color as $\mathcal{C}_c(i)$ are active. Thus, these processors are sampling in their boundary cells (line 7). In the other processors, all interior cells ($c \in \mathcal{C}_i$) of the same color as $\mathcal{C}_c(k)$ are active, where ($k \leftarrow 1$ **to** 4 (or 8 in 3D)). These processors are sampling in their active interior cells (line 13).

Sorting the particle set $\mathcal{P}$ into cell subsets amounts to discretely stratified sampling. To sample from cells, we thus compute the ratio of the particles in the cell to the maximum number of particles in any cell (algorithm 7, line 4). If a uniform random number is less than this ratio, we sample from the particles in that cell using the DRS algorithm (Cardinale, 2013). DRS applies the MH algorithm to particles in the cell.

While algorithm 6 implements one iteration, practical MCMC sampling requires tens of thousands of iterations. The amount of useful work done in each iteration is proportional to the number of moves (accepted or rejected particles).

Figure 4.6 illustrates the communication rounds after sampling boundary cells in each sub-sweep of Fig. 4.5.

Following this strategy, the total communication volume is the same as one full ghost communication, where every processor sends the border information as a ghost layer to neighboring processors and receives the ghost information from them. But, in our strategy, the number of times that processor synchronization has to take place is limited, with a lower bound for



(a) (b) (c) (d)

Figure 4.6: Illustration of the ghost communications after sampling in active boundary cells at each sub-sweep of an iteration.

a larger interior region compared to the border region. This ratio is important to ensure that the computation time exceeds the communication time.

We use this strategy to parallelize the DRS algorithm. In each independent set of the checkerboard decomposition, we run DRS to sample particles. Cardinale (2013) has shown that this algorithm by construction produces a Markov chain that is reversible, aperiodic, and irreducible and presented a proof for that. Using the MH algorithm with an acceptance-rejection scheme (Cardinale, 2013), we can therefore simulate the chain, guaranteeing convergence.

The communication can be improved by taking advantage of remote memory access (RMA) or one-sided communication. This allows an active processor to directly access remote memory on the neighboring processors (active on the interior cells) and "*put*" data there. The program running on the neighbor processor (remote) does not need to call any routines to match the *put* operation (Gropp et al., 2014). This strategy results in better performance due to less synchronization.

4.2.5 Data structure

The choice of data structures is important for the performance of algorithm. In particle methods, we usually store particles in a flat array with n elements. Such a data structure is not appropriate in our parallel approach as it would be expensive to rebuild the array and the cell-lists after each sub-sweep. We instead store particle positions directly in the cells. This enables $\mathcal{O}(1)$ insertion and deletion of particles from each cell, and random access as dictated by the RNG. We therefore use an enumerated set of one vector and one hash map. We use particle positions and particle labels as keys to the hash map. This is done using bitwise operations to combine them into one long integer as the key. For two-dimensional images we have,

```
long key = (long) ParticleLabel;
key      = (key << 32) | y << 16 | x;
```

and for three dimensional images we use,

```
long key = (long) ParticleLabel;
key      = (key << 11) | z;
key      = (key << 22) | y << 11 | x;
```

as the key to the hash map.

Thus, we enforce a limit on the size of each sub-image per processor, which is 65535 × 65535 and 2047 × 2047 × 2047 for two and three-dimensional sub-images, respectively. This is a size limit per processor and does not limit the overall size of the image that is distributed across processors.

Using this data structure, we achieve an overall complexity in $\mathcal{O}(1)$ to access entries randomly, insert, and delete particles.

4.3 Results

Even a slight mistake in the parallelized distribution of DRS leads to biased samples and statistically incorrect results. This provides a sensitive measure to verify correctness of our implementation. We check correctness of the distributed parallel algorithm in comparison with the original reference implementation of Cardinale (2013) on two-dimensional synthetic images. To compare the results, we calculate the pixel-wise L^1 and L^2 distances. We initialize both algorithms identically. Our results of the parallel algorithm on four processors agree with the sequential results of the original algorithm.

This is the very initial results of the correctness of parallel algorithm. Many more further tests are required to provide the validity and correctness of the algorithm. In future, we provide the efficiency of the distributed parallel algorithm both for synthetic and real images and finally we provide the uncertainty of segmentation in 3D light-sheet fluorescence microscopy images.

4.4 SUMMARY

We have presented a distributed-memory parallel implementation of the Discrete Region Sampling (DRS) algorithm (Cardinale, 2013) for uncertainty quantification of image segmentation. Parallelization was made possible by domain decomposition, graph coloring and a lean one-sided communication scheme after each sub-sweep. The parallel strategy resolves the memory issues on the large state spaces of images, but its runtime efficiency remains to be shown.

The algorithm was implemented using the PPM library (Sbalzarini et al., 2006; Awile et al., 2010, 2013) as an efficient middleware for parallel particle-mesh methods.

The parallelization strategy follows the idea of performing many small, computationally cheap moves, but accept only some of them. This is possible for local shape perturbations and energies with limited computational support. Following Cardinale (2013), we used the proposals that locates particles in the neighborhood of the contour and assigns the weights to particles favoring moves towards a smooth contour.

CHAPTER

FIVE

CONCLUSIONS AND FUTURE WORK

We have addressed the problem of acquisition-rate segmentation of large images from fluorescence microscopes by developing a parallel distributed-memory framework. A major challenge in acquisition-rate segmentation of large images are the memory requirements of the image data and the analysis algorithm. Pixel-accurate segmentation is a computationally expensive task that continuously presents performance challenges due to the increasing volume of image data. We addressed both issues by parallelizing a generic, general purpose, model-based image segmentation algorithm, DRC (Cardinale et al., 2012). A distributed-memory parallel implementation of the DRC algorithm for image segmentation developed. Efficient parallelization was made possible by a novel parallel independent sub-graph algorithm, as well as optimizations to the parallel connected-component labeling algorithm. The final algorithm was implemented using the PPM library (Sbalzarini et al., 2006; Awile et al., 2010, 2013) and includes both piecewise constant (PC) and piecewise smooth (PS) image models.

The distributed-memory scalability of the presented approach effectively overcomes the memory and runtime limitations of a single computer. None of the computers or processors over which a task is distributed needs to store the entire image. This allows segmenting very large images. We have shown that a real-world light-sheet microscopy image of $1824 \times 834 \times 809$ pixels (amounting to 4.6 GB of uncompressed memory) was segmented in under 60 seconds when distributed across 128 processors. This was less than the 90 seconds until the microscope acquired the next time point, hence providing online, acquisition-rate image analysis.

We have demonstrated the parallel efficiency and scalability of the present implementation. Correctness of the parallel implementation was shown by comparison with the original sequential reference implementation.

Furthermore, we have developed a novel parallel connected-component labeling algorithm that requires only one round of communication to converge. The presented algorithm scales more consistently than the iterative labeling algorithms of Flanigan and Tamayo (1995) and Moloney and Pruessner (2003) for objects that span across several sub-images.

We have then presented a general algorithm for parallelizing an inherently sequential MCMC algorithm (DRS) and used it to provide a measure of segmentation uncertainty or solution robustness. Our approach applies to energy evaluations of limited computational support, such as piecewise constant and smooth energies, where energy computation does not require global information. The main contribution is the parallelization in a statistically unbiased way that can be executed on many CPU cores. It addresses the memory issues for extremely large state spaces of images.

This thesis, hence addressed the acquisition-rate image segmentation problem and provided a measure of segmentation uncertainty or solution robustness, in multi-dimensional and high-resolution images from modern fluorescence microscopes. We hope that the frameworks and analysis of the parallel DRC and DRS algorithms help researchers use images more effectively and open the door for the smart microscopes of the future.

Despite the encouraging results shown here, there is still room for further performance improvements. One idea could be to compress the particle and image data before communication. This would effectively reduce the communication overhead and improve scalability. Furthermore, spatially adaptive domain decompositions and dynamic load balancing could be used to reduce load imbalance. Depending on the image contents, not all processors may have an equal amount of particles. This causes asynchronous waiting times that may limit scalability. Due to the checkerboard decomposition used in the graph handling of DRC, however, the current implementation is limited to Cartesian domain decompositions, while spatially adaptive trees might be better. The DRS scheme is not limited to any specific domain decomposition.

In the future, we will extend also DRC to spatially adaptive domain decompositions and present the performance. This extension is possible by extending the checkerboard decomposition to a maximal independent sets decomposition, where independent sets can be used for handling the graph.

The local evaluation of energy differences for all possible particle moves can be accelerated by taking advantage of multi-threading and graphics processing units (GPUs). This is possible for DRC, as has already been shown (Ebrahim, 2011), suggesting that processing could be further accelerated by a factor of 10 to 30, depending on the image model.

Due to the independent sets in the parallel MCMC, this strategy is inherently suited for multi-threading and graphics processing units. In future we hence investigate hybrid parallelization strategies for MCMC sampling.

The present parallel algorithm inherits DRC's flexibility and can accommodate for different image models that include prior knowledge about the image-formation process and the imaged objects. In the future, we further investigate this quality for example by including shape or size priors to further improve the segmentation quality.

All presented algorithms can also be further optimized for a specific application, like zebrafish vasculature segmentation. They can then be used to address biological questions and alleviate a major challenge in computational bio-image analysis. This can provide a solution to the big-data problem of multi-dimensional and high-resolution images. The parallelized algorithms are capable of real-time segmentation and uncertainty quantification. Ultimately, we may not need to store the raw data any more, but directly store the analysis results and their associated uncertainties.

Even though DRC provides pixel-accurate segmentation results, it suffers from the user having to adjust many parameters, especially for 3D image segmentation. Generally, the adjustment of these parameters is done by trial and error, and is tedious. Evolutionary algorithms could address parameter adjustments. This can in the future be used to provide assisted parameter tuning or automatic parameter learning for a specific, given image. The presented parallel algorithms also significantly accelerate this tuning process, and uncertainty information from DRS can be used to guide the learning.

BIBLIOGRAPHY

Y. Afshar, F. Schmid, A. Pishevar, and S. Worley, “Exploiting seeding of random number generators for efficient domain decomposition parallelization of dissipative particle dynamics,” *Comp. Phys. Commun.*, vol. 184, no. 4, pp. 1119–1128, 2013.

Y. Al-Kofahi, W. Lassoued, W. Lee, and B. Roysam, “Improved automatic detection and segmentation of cell nuclei in histopathology images,” *IEEE T. Bio-Med. Eng.*, vol. 57, no. 4, pp. 841–852, April 2010.

F. Amat, W. Lemon, D. P. Mossing, K. McDole, Y. Wan, K. Branson, E. W. Myers, and P. J. Keller, “Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data,” *Nat. Methods*, vol. 11, no. 9, pp. 951–958, Sep. 2014.

J. A. Anderson, E. Jankowski, T. L. Grubb, M. Engel, and S. C. Glotzer, “Massively parallel monte carlo for many-particle simulations on gpus,” *J. Comput. Phys.*, vol. 254, pp. 27–38, 2013.

J. A. Anderson, M. E. Irrgang, and S. C. Glotzer, “Scalable metropolis monte carlo for simulation of hard shapes,” *Comp. Phys. Commun.*, vol. 204, pp. 21–30, 2016.

G. Aubert and P. Kornprobst, *Mathematical Problems in Image Processing*, second edition ed., ser. Applied Mathematical Sciences. Springer New York, 2006, vol. 147.

O. Awile, “A domain-specific language and scalable middleware for particle-mesh simulations on heterogeneous parallel computers,” PhD Thesis, Diss. ETH No. 20959, ETH Zürich, 2013.

O. Awile, O. Demirel, and I. F. Sbalzarini, “Toward an object-oriented core of the PPM library,” in *Proc. ICNAAM, Numerical Analysis and Applied Mathematics, International Conference.* AIP, 2010, pp. 1313–1316.

O. Awile, M. Mitrović, S. Reboux, and I. F. Sbalzarini, “A domain-specific programming language for particle simulations on distributed-memory parallel computers,” in *Proc. III Intl. Conf. Particle-based Methods (PARTICLES)*, Stuttgart, Germany, 2013, p. p52.

R. Beare and G. Lehmann, “The watershed transform in ITK - discussion and new developments,” *The Insight Journal*, 06 2006.

S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object recognition using shape contexts,” vol. 24, no. 4, pp. 509–522, 2002.

G. Bertrand, “Simple points, topological numbers and geodesic neighborhoods in cubic grids,” *Pattern. Recogn. Lett.*, vol. 15, no. 10, pp. 1003–1011, 1994.

Y. Boykov and V. Kolmogorov, “Computing geodesics and minimal surfaces via graph cuts,” in *Proc. IEEE Intl. Conf. Computer Vision (ICCV)*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, 2003, pp. 26–33.

P. Bremaud, *Markov chains : Gibbs fields, Monte Carlo simulation and queues.* Springer, 1999, iSBN: 0-387-98509-3.

J. M. R. Byrd, “Parallel markov chain monte carlo,” Ph.D. dissertation, University of Warwick, June 2010.

J. Cardinale, “Unsupervised segmentation and shape posterior estimation under Bayesian image models,” PhD Thesis, Diss. ETH No. 21026, MOSAIC Group, ETH Zürich, 2013.

J. Cardinale, G. Paul, and I. F. Sbalzarini, “Discrete region competition for unknown numbers of connected regions,” *IEEE Trans. Image Process.*, vol. 21, no. 8, pp. 3531–3545, 2012.

V. Caselles, R. Kimmel, and G. Sapiro, “Geodesic active contours,” *Int. J. Comput. Vis.*, vol. 22, no. 1, pp. 61–79, 1997.

J. Chang and J. Fisher, “Efficient topology-controlled sampling of implicit shapes,” in *Image Processing (ICIP), 2012 19th IEEE International Conference on*, 2012, pp. 493–496.

J. Chang and J. W. Fisher III, “Efficient MCMC sampling with implicit shape representations,” in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, IEEE. IEEE, 2011, pp. 2081–2088.

T. F. Coleman and J. J. Moré, “Estimation of sparse jacobian matrices and graph coloring problems,” *SIAM J. Numer. Anal.*, vol. 20, no. 1, pp. 187–209, 1983.

D. Cremers, M. Rousson, and R. Deriche, “A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape,” *Int. J. Comput. Vision.*, vol. 72, no. 2, pp. 195–215, 2007.

A. Delong, A. Osokin, H. N. Isack, and Y. Boykov, “Fast approximate energy minimization with label costs,” *Int. J. Comput. Vision*, vol. 96, no. 1, pp. 1–27, 2012.

R. Diestel, *Graph Theory: Springer Graduate Text GTM 173*, ser. Springer Graduate Texts in Mathematics (GTM). Math. Forschungsinst., 2012.

R. Durstenfeld, “Algorithm 235: Random permutation,” *Commun. ACM*, vol. 7, no. 7, pp. 420–421, Jul. 1964.

E. Ebrahim, “Energy-based image segmentation using GPGPU,” Master thesis, Technische Universität München & MOSAIC Group, ETH Zurich, 2011.

M. Flanigan and P. Tamayo, “Parallel cluster labeling for large-scale monte carlo simulations,” *Physica A: Statistical Mechanics and its Applications*, vol. 215, no. 4, pp. 461–480, 1995.

A. Galizia, D. D’Agostino, and A. Clematis, “An MPI–CUDA library for image processing on HPC architectures,” *J. Comput. Appl. Mech.*, vol. 273, pp. 414–427, 2015.

S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, no. 6, pp. 721–741, 1984.

C. Geyer, "Markov chain Monte Carlo maximum likelihood," in *Computer Science and Statistics, 23th Symposium of the Interface*, April 1991, pp. 156–163.

W. Gropp, T. Hoefler, E. Lusk, and R. Thakur, *Using Advanced MPI: Modern Features of the Message-Passing Interface*, ser. Computer science & intelligent systems. MIT Press, 2014.

W. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

G. S. Heffelfinger and M. E. Lewitt, "A comparison between two massively parallel algorithms for monte carlo computer simulation: An investigation in the grand canonical ensemble," *J. Comput. Chem.*, vol. 17, no. 2, pp. 250–265, 1996.

J. Hoshen and R. Kopelman, "Percolation and cluster distribution. i. cluster multiple labeling technique and critical concentration algorithm," *Phys. Rev. B*, vol. 14, no. 8, pp. 3438–3445, October 1976.

J. Huisken and D. Y. R. Stainier, "Even fluorescence excitation by multidirectional selective plane illumination microscopy (mspim)," *Opt. Lett.*, vol. 32, no. 17, pp. 2608–2610, Sep 2007.

J. Huisken, J. Swoger, F. Del Bene, J. Wittbrodt, and E. H. K. Stelzer, "Optical sectioning deep inside live embryos by selective plane illumination microscopy," *Science*, vol. 305, pp. 1007–1009, 2004.

L. Ibanez, W. Schroeder, L. Ng, and J. Cates, *The ITK Software Guide*, 2nd ed., Kitware, Inc., 2005.

G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Comput. Vis.*, pp. 321–331, 1988.

W. Kim and Y. Kim, "A region-based shape descriptor using Zernike moments," *Signal Processing: Image Communication*, vol. 16, no. 1, pp. 95–102, 2000.

F. Knop and V. Rego, "Parallel labeling of three-dimensional clusters on networks of workstations," *J. Parallel. Distr. Com.*, vol. 49, no. 2, pp. 182–203, 1998.

D. E. Knuth, *The Art of Computer Programming. Sorting and Searching.*, 2nd ed. Addison Wesley, May 1998, vol. 3.

J. Lamy, "Integrating digital topology in image-processing libraries," *Computer Methods and Programs in Biomedicine*, vol. 85, no. 1, pp. 51 – 58, 2007.

C. H. Li and C. K. Lee, "Minimum cross entropy thresholding," *Pattern Recognition*, vol. 26, no. 4, pp. 617–625, 1993.

D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. IEEE Intl. Conf. Computer Vision (ICCV)*, Vancouver, BC, Canada, 2001, pp. 416–423.

M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, pp. 1087–1092, 1953.

N. R. Moloney and G. Pruessner, "Asynchronously parallelized percolation on distributed machines," *Phys. Rev. E*, vol. 67, p. 037701, Mar 2003.

J. Montagnat, H. Delingette, and N. Ayache, "A review of deformable surfaces: topology, geometry and deformation," *Image and Vision Comput.*, vol. 19, no. 14, pp. 1023 – 1040, 2001.

D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Comm. Pure Appl. Math.*, vol. 42, pp. 577–685, 1989.

J. J. K. Ó Ruanaidh and W. J. Fitzgerald, *Numerical Bayesian methods applied to signal processing.* Springer-Verlag New York, 1996.

E. Olmedo, J. D. L. Calleja, A. Benitez, and M. A. Medina, “Point to point processing of digital images using parallel computing,” *International Journal of Computer Science Issues*, vol. 9, no. 3, pp. 1–10, 2012.

R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, “Shape distributions,” *ACM Trans. Graph.*, vol. 21, no. 4, pp. 807–832, 2002.

G. Paul, J. Cardinale, and I. F. Sbalzarini, “Coupling image restoration and segmentation: A generalized linear model/Bregman perspective,” *Int. J. Comput. Vis.*, vol. 104, no. 1, pp. 69–93, 2013.

G. S. Pawley, K. C. Bowler, R. D. Kenway, and D. J. Wallace, “Concurrency and parallelism in MC and MD simulations in physics,” *Comput. Phys. Commun.*, vol. 37, no. 1–3, pp. 251–260, 1985.

P. G. Pitrone, J. Schindelin, L. Stuyvenberg, S. Preibisch, M. Weber, K. W. Eliceiri, J. Huisken, and P. Tomancak, “OpenSPIM: an open-access light-sheet microscopy platform,” *Nat. Methods*, vol. 10, no. 7, pp. 597–598, Jul 2013.

S. Preibisch, S. Saalfeld, J. Schindelin, and P. Tomancak, “Software for bead-based registration of selective plane illumination microscopy data,” *Nat Meth*, vol. 7, no. 6, pp. 418–419, Jun. 2010, 00086.

J.-L. Rose, C. Revol-Muller, D. Charpigny, and C. Odet, “Shape prior criterion based on Tchebichef moments in variational region growing,” in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, Nov. 2009, pp. 1081 –1084.

J. S. Rosenthal, “Parallel computing and monte carlo algorithms,” *Far East Journal of Theoretical Statistics*, vol. 4, pp. 207–236, December 2000.

L. A. Royer, M. Weigert, U. Günther, N. Maghelli, F. Jug, I. F. Sbalzarini, and E. W. Myers, “ClearVolume: open-source live 3D visualization for light-sheet microscopy,” *Nat. Methods*, vol. 12, no. 6, pp. 480–481, Jun. 2015.

I. F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos, “PPM – a highly efficient parallel particle-mesh library for the simulation of continuum systems,” *J. Comput. Phys.*, vol. 215, no. 2, pp. 566–588, 2006.

I. F. Sbalzarini, S. Schneider, and J. Cardinale, “Particle methods enable fast and simple approximation of Sobolev gradients in image segmentation,” *arXiv preprint arXiv:1403.0240v1*, pp. 1–21, 2014.

N. Scherf and J. Huisken, “The smart and gentle microscope,” *Nat. Biotechnol*, vol. 33, no. 8, pp. 815–818, 08 2015.

J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, “Fiji: an open-source platform for biological-image analysis,” *Nat. Methods*, vol. 9, no. 7, pp. 676–682, 2012.

B. Schmid, G. Shah, N. Scherf, M. Weber, K. Thierbach, C. P. Campos, I. Roeder, P. Aanstad, and J. Huisken, “High-speed panoramic light-sheet microscopy reveals global endodermal cell dynamics,” *Nat Commun*, vol. 4, Jul. 2013, 00013.

F. Ségonne, “Segmentation of medical images under topological constraints,” Ph.D. dissertation, Massachusetts Institute of Technology (MIT), December 2005.

Y. Shi and W. C. Karl, “A real-time algorithm for the approximation of level-set-based curve evolution,” *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 645–656, 2008.

A. F. M. Smith and G. O. Roberts, “Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods,” *J. Roy. Stat. Soc. B. Met.*, vol. 55, no. 1, pp. 3–23, 1993.

J. Stegmaier, J. C. Otte, A. Kobitski, A. Bartschat, A. Garcia, G. U. Nienhaus, U. Strähle, and R. Mikut, “Fast segmentation of stained nuclei in terabyte-scale, time resolved 3d microscopy image stacks,” *PLoS ONE*, vol. 9, no. 2, p. e90036, 02 2014.

J. M. Teuler and J. Gimel, “A direct parallel implementation of the hoshen–kopelman algorithm for distributed memory architectures,” *Comp. Phys. Commun.*, vol. 130, no. 1–2, pp. 118–129, 2000.

D. Tiggemann, “Simulation of percolation on massively-parallel computers,” *Int. J. Mod. Phys. C.*, vol. 12, no. 06, p. 871, 2001.

R. Veltkamp and M. Hagedoorn, “4. state of the art in shape matching,” *Principles of visual information retrieval*, p. 87, 2001.

K.-b. Wang, T.-l. Chia, Z. Chen, and D.-c. Lou, “Parallel execution of a connected component labeling operation on a linear array architecture,” *J. Inf. Sci. Eng.*, vol. 19, pp. 353–370, 2003.

G. Winkler, *Image analysis, random fields and Markov chain Monte Carlo methods: A Mathematical Introduction*, 2nd ed. Springer, 2003.

C. Xu, J. Yezzi, A., and J. Prince, “On the relationship between parametric and geometric active contours,” in *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, vol. 1, 29 2000-nov. 1 2000, pp. 483 –489 vol.1.

D. Zhang and G. Lu, “Review of shape representation and description techniques,” *Pattern Recognition*, vol. 37, no. 1, pp. 1–19, 2004.

S. C. Zhu and A. Yuille, “Region competition: Unifying snakes, region growing, and Bayes/MDL for multiband image segmentation,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, no. 9, pp. 884–900, Sep. 1996.

CURRICULUM VITAE

Name:	Yaser Afshar
Born:	April 26th, 1980
Citizen of:	Tehran, Iran

1998 - 2004	B.Sc. Mechanical Engineering at Khaje Nasir Toosi University of Technology, Tehran, Iran
2004 - 2007	M.Sc. Mechanical Engineering at Isfahan University of Technology, Isfahan, Iran
2012 - 2016	Ph.D. studies in Computer Science at Dresden University of Technology, Dresden, Germany. Ph.D. project carried out at the Center for Systems Biology Dresden and the Max Planck Institute of Molecular Cell Biology and Genetics under the supervision of Prof. Dr. Ivo F. Sbalzarini