FACULDADE DE ENGENHARIA DA
UNIVERSIDADE DO PORTO

High-performance computing for acceleration of medical image processing and analysis techniques

Carlos Alex Sander Juvêncio Gulo

Programa Doutoral em Engenharia Informática

Supervisor: João Manuel Ribeiro da Silva Tavares
Faculdade de Engenharia da Universidade do Porto

Co-Supervisor: Antonio Carlos Sementille
Universidade Estadual Paulista "Júlio de Mesquita Filho"

December, 2019

High-performance computing for acceleration of medical image processing and analysis techniques

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor in Informatics Engineering by the Faculty of Engineering of the University of Porto

Carlos Alex Sander Juvêncio Gulo

Master of Computer Science from the Universidade Estadual Paulista "Júlio de Mesquita Filho" (2012)

Specialist degree in Computer Science from the Universidade Estadual de Londrina (2000)

Technologist of Data Processing from the Fundação Educacional de Andradina (1997)

Supervisor

João Manuel Ribeiro da Silva Tavares

Associate Professor with Habilitation of the Mechanical Engineering Department

Faculdade de Engenharia da Universidade do Porto

Co-Supervisor

Antonio Carlos Sementille

Associate Professor of the Computer Science Department

Universidade Estadual Paulista "Júlio de Mesquita Filho"

December, 2019

Acknowledgments

It is a pleasure to thank all people and Institutions that, directly or indirectly, have made this Thesis possible.

I would like to express my gratitude to Prof. João Manuel R. S. Tavares and Prof. Antonio Carlos Sementille, for the opportunity to undertake this research and for their thoughtful guidance throughout the project.

I wish to sincerely thank professors and lecturers at the Doctoral Program in Informatics Engineering, the librarians, and other staff of the Faculdade de Engenharia da Universidade do Porto.

I want to thank all my friends from our lab for providing a pleasant atmosphere. I am indebted to my lab friends for providing a stimulating and fun environment in which I have had happy and memorable moments. All experiences we shared with each other have contributed to our personal growth in some way.

I would also like to thank the Universidade do Estado de Mato Grosso (UNEMAT) of Brazil, and the National Scientific and Technological Development Council "Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq", process 234306/2014-9 grant under reference #2010/15691-0, for the support given.

Lastly, and most importantly, I wish to thank my parents, Armelinda Juvencio Gulo and Jurandir Bento Gulo. They bore me, raised me, supported me, taught me, and loved me. My Ph.D. would not have been successfully finished without support and encouragement from my wife, Tatiana, and my son, Mateus. Earning M.Sc. degree and traveling more than 10 thousand kilometers was an adventure; however, the greatest challenge of our lives came from pursuing this Ph.D. in Portugal. I could not have done it without the continued support and motivation from my wife, my son, family, friends, colleagues, and supervisors.

"The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man."

George B. Shaw

Abstract

The theme addressed in this Ph.D. Thesis combines two huge computing areas, namely high-performance computing and medical image processing and analysis. Techniques for medical image processing and analysis play a crucial role in many clinical scenarios, including diagnosis, follow-up and treatment planning. However, these applications contribute to the generation of vast quantities of data; besides, the high complexity of the algorithms often used are computationally demanding. High-performance computing solutions are adopted, mainly, to reduce the required run-time and achieve real-time.

The combination of these two areas aims to evaluate the computing performance of image processing and analysis algorithms, and then, the adoption of high-performance computing techniques to speed up these algorithms. During this project, new methodologies were developed, implemented, and validated to identify code snippets of image processing and analysis algorithms, that demands high computing power.

This Thesis is organized into two parts: The first part, Part A, introduces the theme, indicates the goals, reports on the work developed, presents the main contributions, and points out the main conclusions and future research perspectives. The second part, Part B, contains four journal articles that were written to report and disseminate the work developed. These articles describe in detail the methodologies and applications briefly introduced in the first part.

The first article in the second part of this Thesis is entitled *Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review*. This work presents an updated systematic review of various techniques of medical image processing and analysis that were accelerated by high-performance computing.

The second article, *Efficient parallelization on GPGPU of an image smoothing method based on a variational model*, describes a new approach for the acceleration of a highly competent algorithm for image noise filtering based on the CUDA architecture.

Finally, the third and fourth articles entitled *Detection of computationally-intensive functions in a medical image segmentation algorithm based on an active contour model* and *Optimizing a medical image registration algorithm based on profiling data towards real-time performing*, respectively, present two well-known medical image processing and analysis algorithms and discuss their acceleration based on a novel computing approach. This new approach applies performance analysis tools commonly available in traditional computer operating systems, without requiring any new setup or developing new performance-measuring techniques and, therefore, ensures the shortest possible learning curve and makes its adoption potentially easy by the community of researchers

from the medical image processing and analysis area.

Keywords: Medical image processing and analysis, Filtering image, Image segmentation, Image registration, High-performance computing, Profiling tools

Resumo

O tema abordado nesta Tese de Doutoramento combina duas grandes áreas da Computação, nomeadamente a computação de alto desempenho e a área de processamento e análise de imagem médica. Técnicas de processamento e análise de imagem desempenham um papel fundamental em muitos cenários clínicos, envolvendo planeamento e seguimento de tratamentos e diagnósticos. Contudo, estas aplicações médicas contribuem na produção de uma enorme quantidade de dados, além disso, a alta complexidade de muitos destes algoritmos requer grande capacidade computacional e tempo de processamento. Soluções de computação de alto desempenho são utilizadas, principalmente, com o objetivo de alcançar o processamento em tempo real.

Especificamente, a união das referidas áreas está relacionada com a avaliação de desempenho computacional de algoritmos de processamento e de análise de imagem, e posteriormente, o emprego de técnicas de computação de alto desempenho para a aceleração dos mesmos. Durante este projeto foram desenvolvidas, implementadas e validadas metodologias para identificar as funções de algoritmos de processamento e análise de imagem que exigem maior esforço computacional.

Esta Tese está organizada em duas partes: a primeira parte, definida como Parte A, introduz o tema, indica os objetivos, descreve resumidamente o trabalho desenvolvido, apresenta as principais contribuições, bem como aponta as conclusões e perspetivas de trabalho futuro. A segunda parte, definida como Parte B, está composta por artigos de revista produzidos para descrever e disseminar o trabalho desenvolvido. Estes artigos descrevem em detalhes as metodologias e aplicações brevemente introduzidas na Parte A.

O primeiro artigo apresentado na Parte B desta Tese, intitulado *Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review*, apresenta uma revisão sistemática da literatura sobre várias técnicas de processamento e análise de imagem médica aceleradas por computação de alto desempenho.

O segundo artigo, *Efficient parallelization on GPGPU of an image smoothing method based on a variational model*, descreve uma nova abordagem baseada em arquitetura CUDA para acelerar o processamento de um algoritmo altamente competente na filtragem de ruído em imagens.

Finalmente, os terceiro e quarto artigos, intitulados *Detection of computationally-intensive functions in a medical image segmentation algorithm based on an active contour model* e *Optimizing a medical image registration algorithm based on profiling data towards real-time performing*, respectivamente, apresentam dois métodos consolidados em processamento e análise de imagem médica e discutem uma nova solução computacional para o processamento de alto desempenho utilizando

técnicas de computação paralela. Esta nova abordagem utiliza ferramentas de análise de desempenho, comumente disponíveis em sistemas operativos computacionais, desta maneira não são necessárias novas configurações, nem desenvolver novas técnicas para avaliação de desempenho. Esta abordagem permite facilitar o uso de técnicas de paralelização pela comunidade de investigadores da área de processamento e análise de imagem médica.

Palavras-chave: Processamento e análise de imagem médica, Suavização de imagem, Segmentação de imagem, Alinhamento de imagem, Computação de alto-desempenho, Ferramentas para *profiling*.

Contents

Part A Thesis report **1**
1 Introduction . . . 3
2 Main objectives . . . 4
3 Organization of the Thesis . . . 5
4 Brief description of the developed work . . . 6
5 Main contributions and accomplishments . . . 8
6 Conclusion and Future research . . . 13
References . . . 15

Part B - Article 1 Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review **23**
Abstract . . . 25
1 Introduction . . . 25
2 Systematic literature review . . . 27
2.1 Review of selected articles . . . 30
3 Discussion . . . 42
4 Conclusion . . . 45
5 Acknowledgments . . . 46
References . . . 46

Part B - Article 2 Efficient parallelization on GPGPU of an image smoothing method based on a variational model **59**
Abstract . . . 61
1 Introduction . . . 61
2 Image smoothing method . . . 63
3 Assessment metrics . . . 65
3.1 Based on intensity error . . . 65
3.2 Based on structural information . . . 66
4 Parallelization of the smoothing method . . . 67
4.1 Setting the occupancy level . . . 68
4.2 Optimizing the memory hierarchy in CUDA . . . 69

4.3 Implementation of kernels in CUDA C 71
5 Experiments and discussion 73
5.1 Test Infrastructure 73
5.2 Results and Discussion 73
6 Conclusions 78
7 Acknowledgments 79
References 79

Part B - Article 3 Detection of computationally-intensive functions in a medical image segmentation algorithm based on an active contour model 85
Abstract 87
1 Introduction 87
2 Background and Related Work 89
2.1 Medical Image Segmentation 89
2.2 Profiling Method 91
2.3 Related Work 94
3 Material and Methods 95
3.1 Experimental Setup 95
3.2 Dataset 96
3.3 Segmentation Results Evaluation 97
3.4 Performance Evaluation 97
4 Results and Discussion 98
4.1 Algorithm Evaluation 98
4.2 Runtime Evaluation 98
4.3 Performance analysis 99
4.4 Effected of the number of used cores 102
5 Conclusion and Future Works 103
6 Acknowledgments 105
References 105

Part B - Article 4 Optimizing a medical image registration algorithm based on profiling data towards real-time performing 111
Abstract 113
1 Introduction 113
2 Background and related works 115
2.1 Medical image registration 116
2.2 Profiling methods 117
2.3 Related works 120
3 Material and Methods 121
3.1 Environment settings 121
3.2 Registration evaluation 122
3.3 Performance evaluation 123
4 Results and discussion 124

4.1 Algorithm Evaluation 124
4.2 Computation time evaluation 125
4.3 Performance analysis 125
5 Conclusions and future research 128
6 Acknowledgments 130
References 130

List of Figures

Part A - Thesis report **3**

1 Parallel GPU-based implementation of the image smoothing method based on a variational model using the CUDA architecture. 10

2 Diagram of the Profiling Method developed to identify time-consuming functions in the Chan-Vese ACM algorithm. 11

3 Call graph representing the most frequently invoked functions in the Chan-Vese algorithm. Function `ReinitPhi` is responsible for locally computing the signed distance function to its zero level set, and was identified by our method as the most called and the most computational time. Function `GetCVC` computes coefficients needed in the Chan-Vese algorithm for the used level set function. Function `Image:data` is used to assign the image element to minimal energy neighborhood; the auxiliary functions `min` and `max` are used in the minimization procedure of the functional with respect to c_1, c_2, and f (more details are available in Article 3 included in Part B). 12

4 Call graph generated by `perf` representing the most often functions called by the FFD nonrigid registration algorithm. Function `reg_getEntropies` is responsible for computing the joint histogram filling. Function `reg_cubic_spline_getDeformationField3D`: generates the deformation field a lattice of equally spaced control points is defined over the reference image using cubic B-splines. Function `ResampleImage3D`: computes the value $I_s(T(x))$ for every pixel x, or voxel in 3D, inside the reference image. In this case, the computational complexity is linearly dependent on the number of pixels/voxels in the reference image. Function `UpdateParameters` measures the quality of a registration using a cost function, such as mutual information. In order to achieve the perfect registration between two images, the parameters of the used transformation are optimized iteratively (more details are available in Article 3 included in Part B). 13

Part B - Article 1 **25**

1 Distribution of selected articles related to techniques of medical image processing and analysis accelerated by high-performance computing solutions published in recent years 43

2 Main parallel programming models applied to accelerate tasks of medical image processing and analysis 45

Part B - Article 2 **61**
1 Original, noisy and smoothed (128x128 pixels) images, respectively . . . 65
2 Representation of the single-instruction multiple-thread model (adapted from [19]) 68
3 Definitions for the settings of each kernel used in the experiments 69
4 Memory spaces accessed by each thread (adapted from [41]) 70
5 Pseudocode of the developed parallel implementation 72
6 Parallel CUDA-based implementation of the adopted image smoothing method 74
7 Processing time of the proposed GPU-based implementation, which scales up exponentially 75
8 Original noisy test image with 4096×4096 pixels and the smoothed images obtained by the CPU- and GPU-based smoothing implementations, respectively 76
9 Original image and the image smoothed by the parallel implementation, respectively 78

Part B - Article 3 **87**
1 Diagram of the Profiling Method. Each part of the diagram shown is described in the text. 92
2 Segmentation of a MR brain image:(a) Original image, (b) Segmentation initializing, (c) Segmentation obtained using the Chan-Vese algorithm. . . 96
3 Call graph generated by `perf` representing the most often functions called by the Chan-Vese algorithm. 101
4 Most time-consuming functions detected by the profiling tools `perf` and `gprof`. 102
5 Time-consuming functions detected by the profiling tools `perf` and `gprof` using OpenMP-based implementation of the Chan-Vese algorithm. 103
6 Means and standard deviations of runtime spent for running the OpenMP-based implementation of the Chan-Vese algorithm. 104

Part B - Article 4 **113**
1 Diagram of the Profiling Method. The function of each stage of the diagram shown is described in the text. 118
2 Call graph generated by `perf` representing the most often called functions in the studied image registration algorithm. 127
3 Proportionality of the time-consumption functions detected by the profiling tools `perf` and `gprof` using the developed OpenMP-based implementation of the FFD algorithm. 129
4 Means and standard deviations of runtime spent for running the developed OpenMP-based implementation of the FFD algorithm under study. 129

List of Tables

Part A - Thesis report **3**

Part B - Article 1 **25**
1 Total number of articles retrieved from each electronic repository 29
2 Total articles retrieved, duplicated and remaining after applying each criteria 29
3 Relevance of each repository used to retrieve articles related to techniques of medical image processing and analysis combined with high-performance computing solutions 30
5 The Impact Factor column was calculated using the ratio of the number of Google citations of the paper and the number of years since its publication. 32

Part B - Article 2 **61**
1 Types of memory access in CUDA [44] 71
2 Comparison between the computational time (in milliseconds) required by the CPU- and GPU-based implementations to smooth the test static images with 50 iterations 73
3 NCC and SSIM values computed for the static test images using 15, 25 and 50 iterations 75
4 PSNR values computed for the static test images before (noisy) and after being smoothed by the CPU- and GPU-based implementations using 15, 25 and 50 iterations 77
5 Frames per second (FPS) rate obtained in the CPU- and GPU-based implementations with the smoothing method applied with 15, 25 and 50 iterations 77

Part B - Article 3 **87**
1 Direct comparison of ground truth and algorithm-based segmentation results for 13 images via the Dice Similarity Coefficient (DSC). 99
2 Means and standard deviations of the runtime (in seconds) required by the sequential-based Chan-Vese algorithm implementation. 99
3 File sizes, indicated in kilo (KB) and megabytes (MB), generated by `perf` according to the images dimension and the OpenMP-based implementation with different number of threads. 100

Part B - Article 4 **113**
1 Comparison of *classical* FFD and profiled-based algorithm results for 13 images based on the Dice Similarity Coefficient (DSC) value. 124
2 Means and standard deviations of the runtime (in seconds) required by the profiled-based FFD's algorithm implementation. 125
3 File sizes, indicated in megabytes (MB), generated by `perf` according to images dimension and the developed OpenMP-based implementation using different number of threads. 126

Part A

Thesis report

1 Introduction

In recent years, important contributions have been made in the field of medical image processing and analysis [1–7] by integrating systems and techniques that support high-level information extraction for purposes of diagnosis, surgical intervention, treatment and follow-up of diseases, as well as optimization of rehabilitation plans [8–11]. The first step in medical imaging consists of acquiring the data using a suitable imaging device and then reconstructing the related images. This information extraction is based on different imaging modalities such as, X-ray [2], computed tomography (CT) [12, 13], magnetic resonance (MR) [14, 15], endoscopy [16], microscopy [17, 18], optical coherence tomography (OCT) [19], functional magnetic resonance (fMR) [10, 20], magnetic resonance elastography (MRE) [21], positron emission tomography (PET) [22–24], single photon emission computed tomography (SPECT) [25], and 3D ultrasound computer tomography (USCT) [6]. After this, many techniques of image processing and analysis can be applied on the acquired images, such as image filtering [19, 26, 27], image segmentation [5, 14, 15] and image registration [9, 13, 28].

Frequently, medical images are corrupted by noise due to the image acquisition procedure or by artifacts generated by data transmission or other processes [29]. In summary, image filtering techniques are applied to remove noise from images so that the processed data can be analyzed more easily using higher-level techniques of computational image analysis, in particular of image segmentation [30] or image registration [31].

The process of image segmentation is widely used in medicine; it is responsible for identifying and delineating interpretable regions within an input image. Frequently, tasks of 3D visualization, interpolation, filtering, and even image registration depend on successful image segmentation in order to achieve better accuracy [32].

At the same direction, image registration is a critical operation performed on medical images in order to establish the spatial correspondence between two or more images acquired by different imaging devices or sensors or/and taken at different angles or time, or/and under different acquisition conditions [17, 33].

However, image segmentation and image registration are both complex tasks and require high processing power to perform and obtain accurate and consistent image-based information; also, there is a need for continuous runtime optimization in order to achieve real-time. Therefore, the use of high-performance computing techniques has attracted considerable interest, and this has allowed operating at high computational speeds with low computational power, particularly in time-constrained applications for medical diagnosis [34]. Consequently, in the last decade, considerable research has

been devoted to the use of techniques of image processing and analysis accelerated by high-performance computing solutions [35–38].

The emerging of multi-core processor architecture provided the development of several studies evolving image filtering, image segmentation, and image registration algorithms on multi-core CPUs [1, 39, 40]. Multi-core architecture was initially being designed for applications fully exploiting the power of this architecture working in parallel; however, writing parallel programs is one of the most complex tasks in computer programming [13, 41]. Fortunately, profiling methods can be used in order to identify performance bottlenecks during the execution of algorithms on CPU under a particular workload; besides, these methods can count the number of times a function is called and display timing information about the function under analysis [42–44]. Therefore, profiling methods can reduce developers' time and labor in code parallelization, especially in legacy algorithms implemented by someone else [43, 45].

2 Main objectives

The main tasks and objectives defined for this Ph.D. project were the following:

- Prepare a literature review of the techniques of medical image processing and analysis that have been accelerated by high-performance computing solutions, as well as identify the metrics used to evaluate computing performance, the parallel designs adopted, and the tasks of medical image processing and analysis involved.
- Development of new computational methodologies for the identification of computationally-intensive functions in medical image processing and analysis algorithms in order to make them suitable for real-time diagnosis by exploiting all the computational power available in typical computer systems.
- The methodologies to be developed should be capable of effectively reducing the processing time of medical image processing and analysis algorithms, regarding tasks of preprocessing, segmentation and registration of medical images.
- The methodologies should be tested and validated using synthetic and real images and evaluated against well-known implementations with different high-performance computing architectures: multi-cores and many-cores.
- The studies to be developed should provide guidelines that can help the researchers of medical image processing and analysis area to detect and evaluate potential

time-consuming functions in their algorithms using profiling tools, facilitating their speed up by exploring techniques of parallel programming.

3 Organization of the Thesis

This Thesis is organized into two main parts: Part A and Part B. This part, Part A, presents the context in which the work was developed, the main objectives defined for this Ph.D. project, a summarized description of the developed tasks, the main contributions achieved, the conclusions and a discussion about possible future works.

Part B is composed of four journal articles written during the development of the Ph.D. project. The articles provide: *1)* a detailed description of the state-of-the-art related to the project; *2)* the parallelization of an image multi-variational filtering algorithm using GPU; *3)* in addition, the identification and acceleration of the time-consuming functions in an image segmentation algorithm: the Chan-Vese active contour model (ACM), and *4)* in an image registration algorithm Free-Form Deformation (FFD) using high-performance computing and profiling tools.

Hence, the outcomes of this project suggest that performance analysis based on profiling can be adapted to effectively reduce the processing times of algorithms, and make them suitable for a real-time diagnosis just by exploiting all the computational power commonly available in modern computers.

The following articles are included in Part B:

Article 1

Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review

Carlos A. S. J. Gulo, Antonio C. Sementille and João Manuel R. S. Tavares

Journal of Real-Time Image Processing (2017), DOI https://doi.org/10.1007/s11554-017-0734-z

Article 2

Efficient parallelization on GPGPU of an image smoothing method based on a variational model

Carlos A. S. J. Gulo, Henrique F. de Arruda, Alex F. de Araújo, Antonio C. Sementille, and João Manuel R. S. Tavares

Journal of Real-Time Image Processing (2016), DOI: https://doi.org/10.1007/s11554-016-0623-x

Article 3

Detection of computationally-intensive functions in a medical image segmentation algorithm based on an active contour model

Carlos A. S. J. Gulo, Antonio C. Sementille and João Manuel R. S. Tavares

Submitted to Journal of Real-Time Image Processing, 2019

Article 4

Optimizing a medical image registration algorithm based on profiling data towards real-time performing

Carlos A. S. J. Gulo, Antonio C. Sementille and João Manuel R. S. Tavares

Submitted to Journal of Medical Systems, 2019

4 Brief description of the developed work

The use of high-performance computing techniques is challenging due to the learning curve required for developers coding medical image applications in parallel design. This Ph.D. project focused on the development of methodologies for detection and evaluation of performance bottleneck snippets[1] in medical image processing and analysis algorithms using profiling models. The proposed approach was applied to a well known medical image segmentation algorithm and also to a well-accepted classical image registration algorithm; besides, a parallelization of an image filtering algorithm based on GPU was also developed. The following works were accomplished to fulfill the objectives established in this project:

- The state-of-the-art performed in this Thesis aimed to present an updated and concise, systematic literature review of the methods already used in the acceleration of techniques of medical image processing and analysis. The most important studies found related to the main high-performance computing methods applied to the acceleration of the techniques of medical image processing and analysis were categorized in terms of the metrics used to evaluate computational performance, the high-performance computing architecture and parallel design involved, and the objects, i.e. tissues or organs, addressed by the techniques. The advantages and limitations of each study were also discussed. Additionally, the review evidenced

[1] Snippets is a common known programming term for a small block of re-usable source code, machine code or text.

the most significant programming efforts developed in the reviewed studies: the learning curve required for parallel programming implementations in order to attain a complete understanding of the advanced concepts related to memory hierarchy, and the design of the shortest-possible optimal data paths. The review is presented in Article 1 included in Part B.

- Image filtering methods have attracted much attention in recent years. Ultrasound is a non-invasive, low-cost imaging modality that has proved useful for many medical applications. However, in practice, ultrasound images are corrupted by speckle noise, which complicates image processing tasks such as tissue segmentation. Many of the original images that need to be enhanced have large dimensions and need to be processed in real time. Therefore, the use of parallel computing strategies has attracted attention, and this has led to higher processing speeds, particularly in time-constrained applications for medical diagnosis. The method developed by Jin and Yang [46] for smoothing of images corrupted by multiplicative noise was accelerated based on general purpose computing on graphics processing units techniques. This new approach is based on compute unified device architecture (CUDA), which is focused on massively parallel programming; thus, the multiplicative noise smoothing method can operate with parallelization strategies based on the data decomposition technique. In the proposed approach, the input image data are stored in the GPU's memory in order to reduce as many data accesses as possible to the main memory system. Thus, input image processing is executed in parallel in the GPU. The proposed method achieved noise smoothing in real time, reducing the runtime by up to 10.65 times compared with the CPU-based implementation, for a set of six images with a different dimension. The new method is presented in Article 2 included in Part B.

- The adoption of high-performance computing techniques by image processing and analysis community is an intensive focus of research since it plays an essential role in the treatment and follow-up of diseases in real-time. Image segmentation is one of the most critical operations performed on medical images; however, these operations require developing optimization strategies in order to reduce runtime. Thus, an approach to identify the most time-consuming functions in the well-known image segmentation algorithm named Chan-Vese [47] was developed. The novel profiling model applies performance analysis tools commonly available in traditional computing operating system. The overall cost of execution time, memory accesses, and performance bottlenecks are measured in execution time.

A call graph visualization can suggest to users a quick graphical overview of the execution time of their codes and, therefore, guarantees the shortest possible learning curve by the community of researchers from medical image processing and analysis. The method is described and discussed in Article 3 included in Part B.

- Image registration is a critical operation performed on medical images. The free-form deformation algorithm [48, 49] is a well-established technique developed to perform nonrigid image registration; however, it is incredibly time-consuming. Hence, the developed approach for detecting the potential parallelism in order to exploit all the computational power commonly available in modern computers was used again on this algorithm. The applied approach significantly reduces the difficulty of measuring programs concerning the widely used technique of placing some code on each basic block of the algorithm under study. Guidelines were developed aiming to help the community of researchers from medical image processing and analysis to achieve real-time in nonrigid image registration applications. The multi-thread parallelization implemented using the application programming interface OpenMP of the most costly functions in the free-form deformation algorithm reduced the runtime by up to 7 times compared with the single thread-based implementation. The new approach is presented in Article 4 included in Part B.

5 Main contributions and accomplishments

The research in this Thesis focused on developing methodologies for reducing time to implement parallel versions of medical image processing and analysis algorithms. The most important contributions of this Ph.D. project were the following:

- A comprehensive systematic literature review of techniques of medical image processing and analysis that have been accelerated by different high-performance computing solutions was successfully reviewed. The content contained in this review is important to provide a better understanding of methods, techniques, imaging modalities, metrics of computational performance, and the most frequently used computing architectures.

  This concise and up-to-date review reveals that the most significant programming efforts found in parallel programming model are: *a)* the learning curve required for programming parallel implementations, *b)* obtaining a complete understanding

of the advanced concepts related to memory hierarchy, and *c)* the design of the shortest-possible optimal data paths. Additionally, there is another aspect point out by the reviewed authors: modifying the design of a CPU-based algorithm in order to make it parallel usually requires changing the programming model, the programming language, and the memory strategy. Thus, this review contributed for evidencing a potential topic for further research. Moreover, the methods described in this review are classified in terms of the parallel computing technique used in filtering, segmenting, and in registering different medical images. Additionally, the review can help other researchers to exploit the benefits of parallelization and improve their application performance for routine clinical use in real-time. The developed review was disseminated through Article 1 included in Part B.

- Development of a new optimization of the multi-variational image noise smoothing algorithm [46] based on general purpose computing on graphics processing units (GPU) techniques. Microscopy, ultrasound and infrared images are typically corrupted by multiplicative noise during the acquisition procedure or by artifacts generated by data transmission. An image smoothing method based on a variational model has been successfully applied to medical images; however, at a high computational cost, especially for high dimensional images. The computational complexity of this algorithm for processing input images is equal to $O(m \times n \times T)$, where m and n are the number of rows and columns of the input image, which are processed for T iterations. Our approach using GPU achieved high processing performance at a low cost when compared with sequential implementations or parallel implementations in multi-computers. The whole input image processing is executed in parallel in the GPU, beginning with the memory allocation in the device's memory (GDRAM) and then copying the input image as a data matrix from the memory system (RAM) to the device's memory. The parallel implementation has transparent and portable scalability in GPUs; the developed image smoothing method based on the variational model using CUDA architecture is illustrated in the Figure 1 and explained in details in Article 2 included in Part B.

- Development of a novel model for detecting computationally-intensive functions in the well known Chan-Vese image segmentation algorithm. This algorithm has been used in contour detection through a topological change of the segmentation curves, mainly by medical image processing and analysis community, and has already been validated by various numerical results. The pseudo-code of the Chan-Vese algorithm is presented in Algorithm 1:

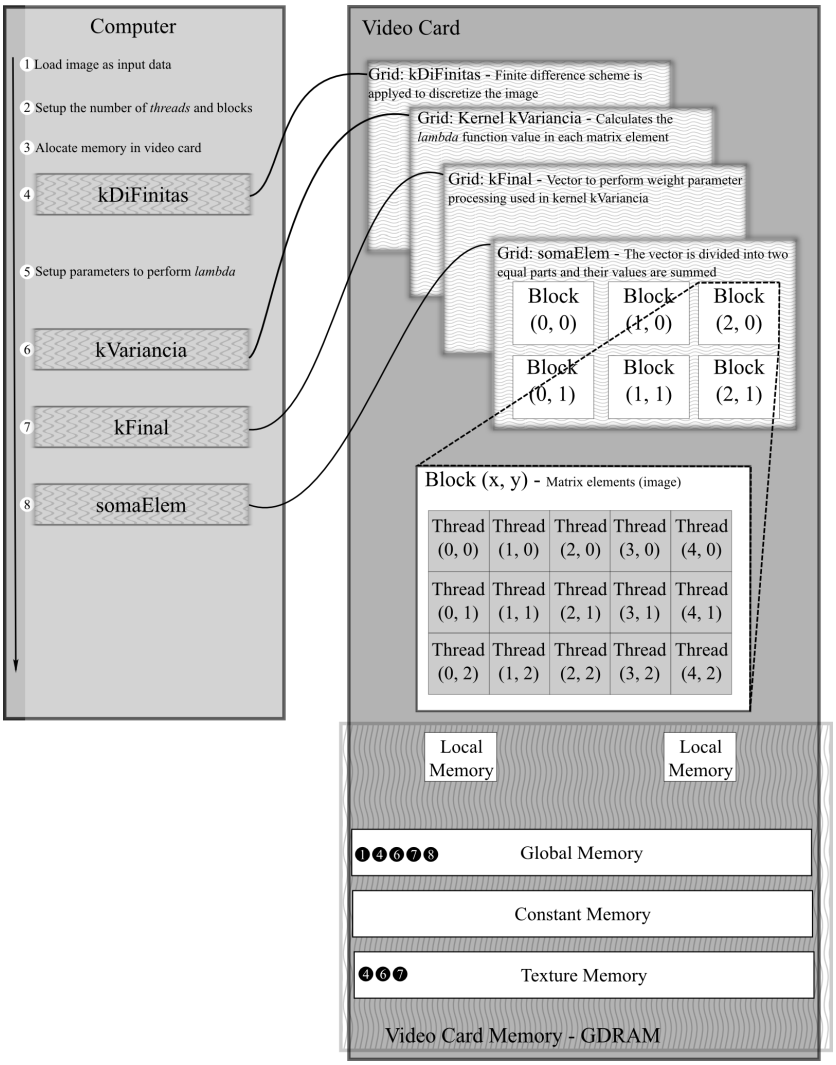


Figure 1: Parallel GPU-based implementation of the image smoothing method based on a variational model using the CUDA architecture.

Our approach, shown in Figure 2, applies performance analysis tools to measure the algorithm performance concerning the overall cost of execution time, memory access, and performance bottlenecks. For measuring the performance of each function of the Chan-Vese algorithm implementation, we focused on gathering profile tools for collecting data while monitoring software interruptions and performance counters. Then, the gathered data is analyzed to extract performance statistics and also record the arc in the call graph for activating the function under analysis. The generated call graph is then analyzed, regarding time-consuming functions and the number of times these functions were invoked. In the Chan-Vese algorithm, the function responsible for computing locally the signed distance function to its zero level set was the most frequently called and the most costly one,

Algorithm 1: Chan-Vese Segmentation Algorithm

Input: Image $I(x,y)$
Preprocessing;
    Compute feature map (Input Image I);
    Compute gradient map $G = \nabla G_\sigma \oplus I$;
    Normalize G;
    Compute regional information-based normalized feature map F;
Initialize φ;
for $n = 1, 2, \ldots, N_{max}$ **do**
    Search the 3 x 3 neighborhood;
    Compute c_1 and c_2 as the region averages;
    Evolve φ with one semi-implicit timestep;
    if $\|\varphi^{n+1}$ - $\varphi^n\|_2 / |\Omega| \leq tol$ **then**
        stop;
    end
end
Update the contour information;

as shown in Figure 3. Afterward, a high-performance computing implementation was developed in order to effectively reduce the runtime of the Chan-Vese image segmentation method, making it suitable for real-time segmentation by simply exploiting all the computational power available in a common personal computer.

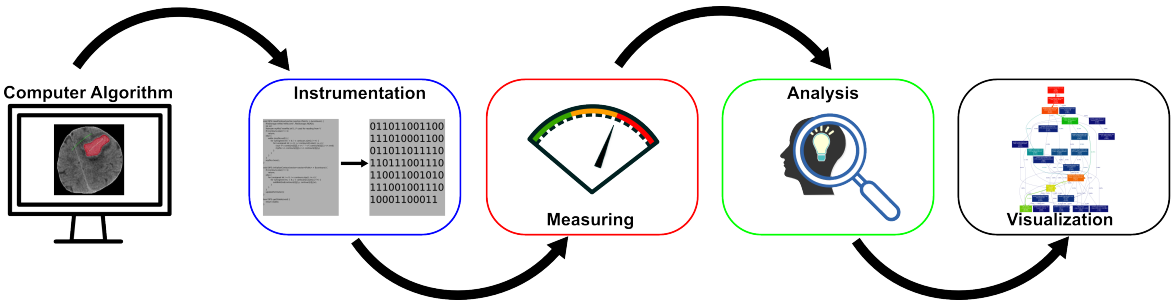


Figure 2: Diagram of the Profiling Method developed to identify time-consuming functions in the Chan-Vese ACM algorithm.

Parallelization assisted by our method combined an approach to detect the available parallelism in an algorithm and also substantially reduced the overall time needed for writing parallel implementations from scratch.

- The method previously developed was applied for identifying potential parallelism and evaluate possible optimization snippets, using the support call graph visualization provided by performance analysis tools, in a nonrigid image registration algorithm: the Free-Form Deformation (FFD) algorithm [48, 49],

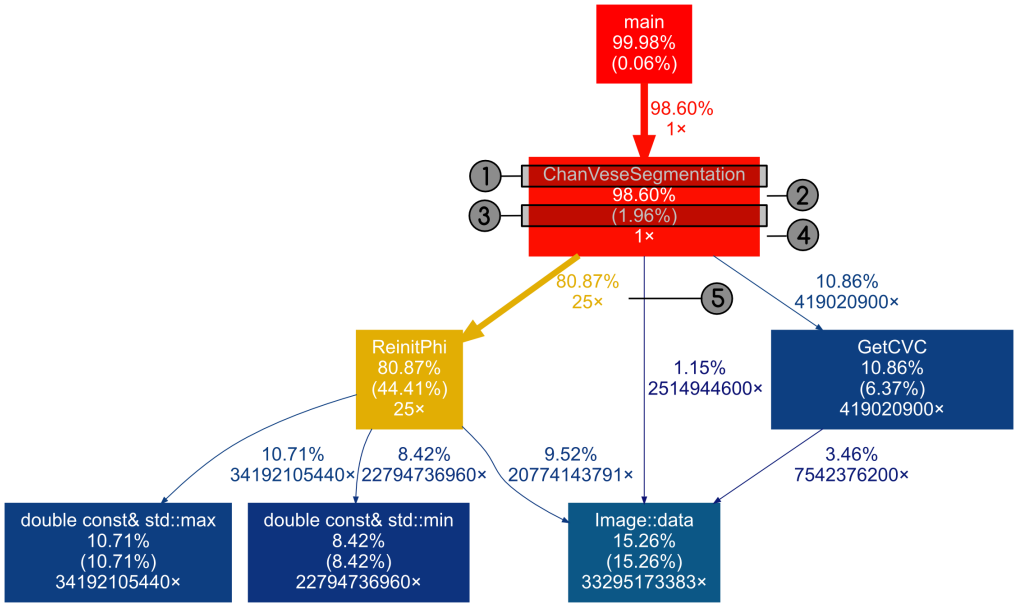


Figure 3: Call graph representing the most frequently invoked functions in the Chan-Vese algorithm. Function `ReinitPhi` is responsible for locally computing the signed distance function to its zero level set, and was identified by our method as the most called and the most computational time. Function `GetCVC` computes coefficients needed in the Chan-Vese algorithm for the used level set function. Function `Image:data` is used to assign the image element to minimal energy neighborhood; the auxiliary functions `min` and `max` are used in the minimization procedure of the functional with respect to c_1, c_2, and f (more details are available in Article 3 included in Part B).

which is one of the most popular image registration algorithms used in medical applications. The computations of the geometric transformation and the similarity measure were the time-consuming bottlenecks identified in the studied FFD registration algorithm. Basically, the approach presented in Figure 2 was also used with the FFD registration algorithm; the only modification was concerning the monitoring runtime behavior of the algorithm implementation under analysis, which involves aggregating information on the base of the number executions of every basic-block, instrumenting different type of events, such as *free* and *malloc*, and similar low level functions.

The studied FFD nonrigid registration algorithm is computationally costly, and to accelerate it, it should be taken into account the transformation of the floating image using splines and an interpolation function, the evaluation of an objective function, besides the optimization of this function. The generated call graph is shown in Figure 4, representing the time propagated for each function from its descendants, and the number of times each function was called.

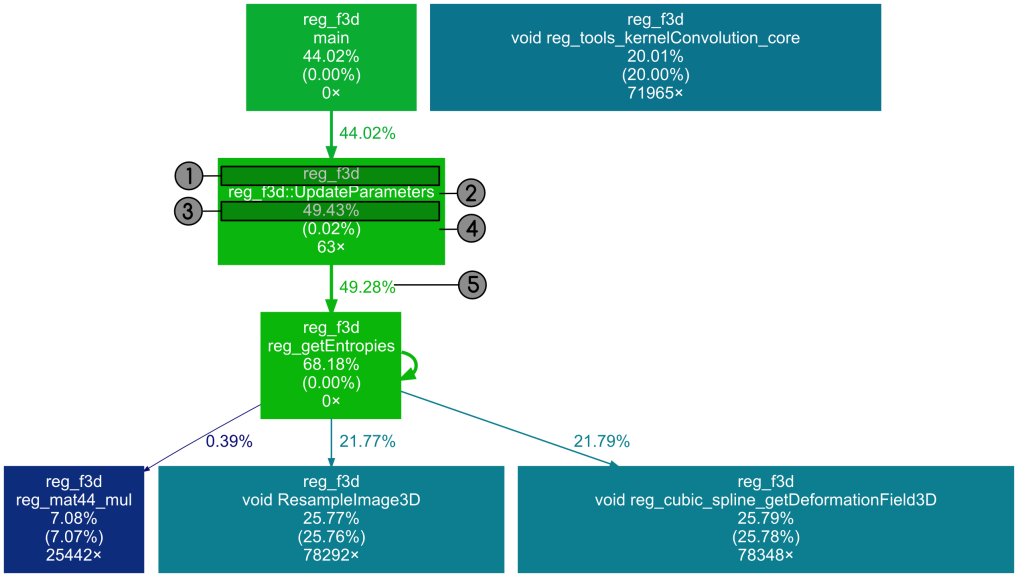


Figure 4: Call graph generated by `perf` representing the most often functions called by the FFD nonrigid registration algorithm. Function `reg_getEntropies` is responsible for computing the joint histogram filling. Function `reg_cubic_spline_getDeformationField3D`: generates the deformation field a lattice of equally spaced control points is defined over the reference image using cubic B-splines. Function `ResampleImage3D`: computes the value $I_s(T(x))$ for every pixel x, or voxel in 3D, inside the reference image. In this case, the computational complexity is linearly dependent on the number of pixels/voxels in the reference image. Function `UpdateParameters` measures the quality of a registration using a cost function, such as mutual information. In order to achieve the perfect registration between two images, the parameters of the used transformation are optimized iteratively (more details are available in Article 3 included in Part B).

> This case study showed that profiling tools could assist programmers to quickly identify the critical bottlenecks in their algorithms and then develop their parallel implementation.

To the best of our knowledge, no other study suggested profiling tools to support and facilitate the parallelization of the Chan-Vese image segmentation algorithm neither of the FFD nonrigid image registration algorithm.

Based on the contributions achieved during this project, four articles were written and submitted to international journals.

6 Conclusion and Future research

The deployment of high-performance computing techniques is critical in many clinical scenarios to reduce the runtime of algorithms used for medical image processing and

analysis.

The main contributions of this Ph.D. project were the development of strategies to exploit the computational power commonly available in traditional computing systems such as multi-core and GPU. Our GPU-based parallelization approach of the image smoothing variational model is portable, transparent, and scalable due to the parallel design adopted in the implementation [50–52]. Developing parallel algorithms assisted by our profiling method does not require any new computing system, can increase the application performance and facilitate implementation efforts, which makes its adoption potentially easy by the community of researchers from medical image processing and analysis.

The methodologies developed for acceleration of the Chan-Vese and FDD algorithms were successfully evaluated using real images, always maintaining the accuracy and robustness of the original algorithm, as can be confirmed in Articles 3 and 4 of Part B of this Thesis.

As a conclusion of this Ph.D., the initial goals were successfully reached, which is confirmed by the fact that the studies developed were published in international journals.

Although the proposed methods were shown to be effective in the acceleration of the filtering, segmentation, and registration of medical images, it is recognized that there are limitations that can be tackled by the following future research:

- Extend the image smoothing GPU-based implementation to perform in multi-GPUs, besides combining it with multi-thread (OpenMP) and multicomputer (MPI); then, optimize this heterogeneous purpose of achieving higher performances regarding runtime.

- Optimize the most time-consuming functions already detected in the image smoothing algorithm by using heterogeneous parallel computing platforms based on GPUs. Algorithm parallelization assisted by profiling tools would not only increase the maximum application performance and reduce the required implementation efforts but also provide a pragmatic incentive for developers to begin exploiting this strategy.

- The most time-consuming functions detected in the image segmentation and registration algorithms, respectively described in Articles 3 and 4 of Part B, can be optimized, and further speedups can be achieved using more sophisticated data-parallel algorithms. Optimizing these algorithms by using heterogeneous parallel computing platforms based on GPUs is also recommended. Additional challenges can be addressed; for instance, the issue in shared memory systems

for protecting simultaneous data access in order to avoid data inconsistency and errors, load balancing, and the efficient management of reading/writing data on the mass storage devices. These challenging requirements are all critical for achieving efficiency and the maximum performance possible in the used computer architecture. Both image segmentation and registration optimized algorithms have been evaluated using only one image dataset; therefore, it is always recommended to evaluate our implementations using different datasets and different evaluation criteria.

References

[1] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. A survey of medical image registration on multicore and the GPU. *IEEE Signal Processing Magazine*, 27(2): 50–60, 2010. ISSN 1053-5888. doi: 10.1109/MSP.2009.935387.

[2] J. Treibig, G. Hager, H. Hofmann, J. Hornegger, and G. Wellein. Pushing the limits for medical image reconstruction on recent standard multicore processors. *International Journal of High Performance Computing Applications*, 27(2): 162–177, 2013. doi: 10.1177/1094342012442424.

[3] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte. Medical image processing on the GPU - past, present and future. *Medical Image Analysis*, 17(8):1073–1094, DEC 2013. ISSN 1361-8415. doi: 10.1016/j.media.2013.05.008.

[4] Y. Zhuge, Y. Cao, J. K. Udupa, and R. W. Miller. Parallel fuzzy connected image segmentation on GPU. *Medical Physics*, 38(7):4365–4371, JUL 2011. ISSN 0094-2405. doi: 10.1118/1.3599725.

[5] W. Shi, Y. Li, Y. Miao, and Y. Hu. Research on the key technology of image guided surgery. *Przeglad Elektrotechniczny*, 88(3B):29–33, 2012. ISSN 0033-2097.

[6] M. Birk, R. Dapp, N. Ruiter, and J. Becker. GPU-based iterative transmission reconstruction in 3D ultrasound computer tomography. *Journal of Parallel and Distributed Computing*, 74(1):1730–1743, 2014. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2013.09.007.

[7] R. Mafi and S. Sirouspour. GPU-based acceleration of computations in nonlinear finite element deformation analysis. *International Journal for Numerical Methods in Biomedical Engineering*, 30(3):365–381, 2014. doi: 10.1002/cnm.2607.

[8] W. Higgins and R. Swift. Distributed system for processing 3D medical images. *Computers in Biology and Medicine*, 27(2):97–115, MAR 1997. ISSN 0010-4825. doi: 10.1016/S0010-4825(96)00042-X.

[9] S. Warfield, F. Jolesz, and R. Kikinis. A high performance computing approach to the registration of medical imaging data. *Parallel Computing*, 24(9-10):1345–1368, 1998.

[10] J.-Y. Yeh and J. Fu. Parallel adaptive simulated annealing for computer-aided measurement in functional MRI analysis. *Expert Systems with Applications*, 33 (3):706–715, 2007. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2006.06.018.

[11] E. Gabriel, V. Venkatesan, and S. Shah. Towards high performance cell segmentation in multispectral fine needle aspiration cytology of thyroid lesions. *Computer Methods and Programs in Biomedicine*, 98(3):231–240, 2010. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2009.07.008.

[12] M. Miller and C. Butler. 3D maximum a posteriori estimation for single photon emission computed tomography on massively-parallel computers. *IEEE Transactions on Medical Imaging*, 12(3):560–565, Sep 1993. ISSN 0278-0062. doi: 10.1109/42.241884.

[13] N. D. Ellingwood, Y. Yin, M. Smith, and C.-L. Lin. Efficient methods for implementation of multi-level nonrigid mass-preserving image registration on GPUs and multi-threaded CPUs. *Computer Methods and Programs in Biomedicine*, 127: 290 – 300, 2016. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2015.12.018.

[14] T. Daggett and I. Greenshields. Parallelization of classification algorithms for medical imaging on a cluster computing system. In *11TH IEEE Symposium on Computer-Based Medical Systems, Proceedings*, pages 305–310. IEEE Comp Soc; IEEE Comp Soc Tech Comm Computat Med; IEEE S Plains Sect-Reg V; Int Soc Optical Engn (SPIE); TX Tech Univ Health Sci Ctr, Dept Radiol, 1998. ISBN 0-8186-8563-8. doi: 10.1109/CBMS.1998.701384.

[15] N. Aitali, B. Cherradi, A. E. Abbassi, O. Bouattane, and M. Youssfi. Parallel implementation of bias field correction fuzzy c-means algorithm for image segmentation. *International Journal of Advanced Computer Science and Applications*, 7(3):375–383, 2016. ISSN 2158-107X.

[16] R. Melo, G. Falcao, and J. Barreto. Real-time HD image distortion correction in heterogeneous parallel computing systems using efficient memory access patterns. *Journal of Real-Time Image Processing*, 11(1):83–91, 2016. doi: 10.1007/s11554-012-0304-3.

[17] T. Rohlfing and J. Maurer, C.R. Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees. *IEEE Transactions on Information Technology in Biomedicine*, 7(1):16–25, 2003. ISSN 1089-7771. doi: 10.1109/TITB.2003.808506.

[18] V. Kumar, B. Rutt, T. Kurc, U. Catalyurek, T. Pan, S. Chow, S. Lamont, M. Martone, and J. Saltz. Large-scale biomedical image analysis in Grid environments. *IEEE Transactions on Information Technology in Biomedicine*, 12(2):154–161, 2008. ISSN 1089-7771. doi: 10.1109/TITB.2007.908466.

[19] P. Rodrigues and R. Bernardes. 3-D adaptive nonlinear complex-diffusion despeckling filter. *IEEE Transactions on Medical Imaging*, 31(12):2205–2212, DEC 2012. ISSN 0278-0062. doi: 10.1109/TMI.2012.2211609.

[20] D. Akgun, U. Sakoglu, J. Esquivel, B. Adinoff, and M. Mete. GPU accelerated dynamic functional connectivity analysis for functional MRI data. *Computerized Medical Imaging and Graphics*, 43:53 – 63, 2015. ISSN 0895-6111. doi: http://dx.doi.org/10.1016/j.compmedimag.2015.02.009.

[21] M. Doyley, E. Van Houten, J. Weaver, S. Poplack, L. Duncan, F. Kennedy, and K. Paulsen. Shear modulus estimation using parallelized partial volumetric reconstruction. *IEEE Transactions on Medical Imaging*, 23(11):1404–1416, 2004. ISSN 0278-0062. doi: 10.1109/TMI.2004.834624.

[22] O. Dandekar and R. Shekhar. FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions. *IEEE Transactions on Biomedical Circuits and Systems*, 1(2):116–127, 2007. ISSN 1932-4545. doi: 10.1109/TBCAS.2007.909023.

[23] P. Kegel, M. Schellmann, and S. Gorlatch. Using OpenMP vs. threading building blocks for medical imaging on multi-cores. 5704 LNCS:654–665, 2009. doi: 10.1007/978-3-642-03869-3_62.

[24] P. Kegel, M. Schellmann, and S. Gorlatch. Comparing programming models for medical imaging on multi-core systems. *Concurrency and Computation-Practice*

& Experience, 23(10):1051–1065, JUL 2011. ISSN 1532-0626. doi: 10.1002/cpe.1671.

[25] A. Formiconi, A. Passeri, M. Guelfi, M. Masoni, A. Pupi, U. Meldolesi, P. Malfetti, L. Calori, and A. Guidazzoli. World wide web interface for advanced spect reconstruction algorithms implemented on a remote massively parallel computer. *International Journal of Medical Informatics*, 47(1-2):125–138, 1997. doi: 10.1016/S1386-5056(97)00089-0.

[26] T.-A. Nguyena, A. Nakib, and H.-N. Nguyen. Medical image denoising via optimal implementation of non-local means on hybrid parallel architecture. *Computer Methods and Programs in Biomedicine*, 129:29 – 39, 2016. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2016.02.002.

[27] C. A. S. J. Gulo, H. F. de Arruda, A. F. de Araujo, A. C. Sementille, and J. M. R. S. Tavares. Efficient parallelization on GPU of an image smoothing method based on a variational model. *Journal of Real-Time Image Processing*, Jul 2016. ISSN 1861-8219. doi: 10.1007/s11554-016-0623-x.

[28] J. Rohrer and L. Gong. Accelerating 3D nonrigid registration using the cell broadband engine processor. *IBM Journal of Research and Development*, 53(5), 2009.

[29] E. López-Rubio. Restoration of images corrupted by Gaussian and uniform impulsive noise. *Pattern Recogn.*, 43(5):1835–1846, 2010.

[30] Z. Ma, R. N. M. Jorge, and J. M. R. S. Tavares. A shape guided C-V model to segment the levator ani muscle in axial magnetic resonance images. *Med. Eng. Phys.*, 32(7):766–774, 2010.

[31] F. P. M. Oliveira, T. C. Pataky, and J. M. R. S. Tavares. Registration of pedobarographic image data in the frequency domain. *Comput. Methods Biomech. Biomed. Eng.*, 3(6):731–740, 2010.

[32] J. Duan, Z. Pan, X. Yin, W. Wei, and G. Wang. Some fast projection methods based on Chan-Vese model for image segmentation. *EURASIP Journal on Image and Video Processing*, 2014(1):7, Jan 2014. ISSN 1687-5281. doi: 10.1186/1687-5281-2014-7. URL `https://doi.org/10.1186/1687-5281-2014-7`.

[33] T. Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum. 3D nonrigid registration via optimal mass transport on the GPU. *Medical Image Analysis*, 13(6): 931–940, 2009. ISSN 1361-8415. doi: http://dx.doi.org/10.1016/j.media.2008.10.008.

[34] A. Merigot and A. Petrosino. Parallel processing for image and video processing: issues and challenges. *Parallel Comput.*, 34(12):694–699, September 2008.

[35] R. Aspin, M. Smith, C. Hutchinson, and L. Funk. MediVol: An initial study into real-time, interactive 3D visualisation of soft tissue pathologies. In D. Roberts, A. ElSaddik, and A. Ferscha, editors, *DS-RT 2008: 12TH 2008 IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, Proceedings*, IEEE ACM International Symposium on Distributed Simulation and Real-Time Applications, pages 103+. IEEE; ACM; IEEE Comp Soc, Tech Comm Parallel Proc; IEEE Comp Soc,Tecn Comm Simulat; IEEE Comp Soc,Tech Comm Comp Architect; ACM SIGSIM, 2008. ISBN 978-0-7695-3425-1. doi: 10.1109/DS-RT.2008.9.

[36] H. Ai and Y. Yu. A distributed parallel processing method for ortho-rectifying satellite imagery. volume 7497, 2009. doi: 10.1117/12.832462.

[37] M. R. Bosisio, J.-M. Hasquenoph, L. Sandrin, P. Laugier, S. L. Bridal, and S. Yon. Real-time chirp-coded imaging with a programmable ultrasound biomicroscope. *IEEE Transactions on Biomedical Engineering*, 57(3):654–664, MAR 2010. ISSN 0018-9294. doi: 10.1109/TBME.2009.2033036.

[38] V. Archirapatkave, H. Sumilo, S. See, and T. Achalakul. Gpgpu acceleration algorithm for medical image reconstruction. pages 41–46, 2011. doi: 10.1109/ISPA.2011.18.

[39] R. Palomar, J. Gómez-Luna, F. A. Cheikh, J. Olivares-Bueno, and O. J. Elle. High-performance computation of bézier surfaces on parallel and heterogeneous platforms. *International Journal of Parallel Programming*, May 2017. ISSN 1573-7640. doi: 10.1007/s10766-017-0506-1. URL `https://doi.org/10.1007/s10766-017-0506-1`.

[40] J. Shackleford, N. Kandasamy, and G. Sharp. *High Performance Deformable Image Registration Algorithms for Manycore Processors*. Morgan Kaufmann Publishers Inc., 2013. ISBN 0124077412, 9780124077416. doi: https://doi.org/10.1016/B978-0-12-407741-6.00007-4.

[41] S. L. Graham, P. B. Kessler, and M. K. McKusick. gprof: A call graph execution profiler. *ACM SIGPLAN Notes*, 39(4):49–57, April 2004. ISSN 0362-1340. doi: 10.1145/989393.989401. URL http://doi.acm.org/10.1145/989393.989401.

[42] M. Dimakopoulou, S. Eranian, N. Koziris, and N. Bambos. Reliable and efficient performance monitoring in Linux. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-4673-8815-3. URL http://dl.acm.org/citation.cfm?id=3014904.3014950.

[43] C. Li, S. Balla-Arabé, and F. Yang. Embedded multi-spectral image processing for real-time medical application. *Journal of Systems Architecture*, 64:26–36, 2016. doi: 10.1016/j.sysarc.2015.12.002.

[44] M. Schulz and B. R. de Supinski. Practical differential profiling. In *Euro-Par 2007 Parallel Processing*, pages 97–106. Lecture Notes in Computer Science, Springer, 2007. ISBN 978-3-540-74466-5. doi: 10.1007/978-3-540-74466-5_12. URL http://dx.doi.org/10.1007/978-3-540-74466-5_12.

[45] S. Rul, H. Vandierendonck, and K. D. Bosschere. A profile-based tool for finding pipeline parallelism in sequential programs. *Parallel Computing*, 36(9): 531 – 551, 2010. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2010.05.006. URL http://www.sciencedirect.com/science/article/pii/S0167819110000840.

[46] Z. Jin and X. Yang. A variational model to remove the multiplicative noise in ultrasound images. *J. Math. Imaging Vis.*, 39(1):62–74, 2011.

[47] T. Chan and L. Vese. An active contour model without edges. In M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, editors, *Scale-Space Theories in Computer Vision: Second International Conference*, volume 1682 of *Lecture Notes in Computer Science*, pages 141–151, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48236-9. doi: 10.1007/3-540-48236-9_13. URL https://doi.org/10.1007/3-540-48236-9_13.

[48] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Nonrigid registration using free-form deformations: application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8):712–721, Aug 1999. ISSN 0278-0062. doi: 10.1109/42.796284.

[49] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, and S. Ourselin. Fast free-form deformation using graphics processing units. *Computer Methods and Programs in Biomedicine*, 98(3):278 – 284, 2010. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2009.09.002. URL `http://www.sciencedirect.com/science/article/pii/S0169260709002533`. HP-MICCAI 2008.

[50] D. Kirk and W.-M. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, 2010. ISBN 978-0-12-381472-2.

[51] W. M. Hwu. *GPU Computing GEMS*. Emerald ed. Morgan Kaufmann and NVIDIA, 2011. ISBN 978-0-12-384988-5.

[52] N. Wilt. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley, Reading, 2013.

Part B - Article 1

Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review

Abstract

Techniques of medical image processing and analysis play a crucial role in many clinical scenarios, including in diagnosis and treatment planning. However, immense quantities of data and high complexity of the algorithms often used are computationally demanding. As a result, there now exists a wide range of techniques of medical image processing and analysis that require the application of high-performance computing solutions in order to reduce the required runtime. The main purpose of this review is to provide a comprehensive reference source of techniques of medical image processing and analysis that have been accelerated by high-performance computing solutions. With this in mind, the articles available in the Scopus and Web of Science electronic repositories were searched. Subsequently, the most relevant articles found were individually analyzed in order to identify: (a) the metrics used to evaluate computing performance, (b) the high-performance computing solution used, (c) the parallel design adopted, and (d) the task of medical image processing and analysis involved. Hence, the techniques of medical image processing and analysis found were identified, reviewed, and discussed, particularly in terms of computational performance. Consequently, the techniques reviewed herein present the progress made so far in reducing the computational runtime involved, and the difficulties and challenges that remain to be overcome.

Keywords: Medical imaging, Image segmentation, Image registration, Image reconstruction

1 Introduction

Throughout the history of computer systems, the evolution of processors and increases in computing speed have been closely related. Traditionally, the integrated circuit industry has fitted ever more transistors into a single chip thereby achieving high performance [1]. However, this approach is limited by physical restrictions of silicon, mainly excessive energy consumption and overheating of processors [2].

In recent years, advances in this area have taken a different direction, leading to modern processor architecture used for (a) multi-core CPUs (which contain two or more processing cores) and (b) the general purpose computing on graphics processing units (GPGPU), which is defined in this review as “many-core architecture”. Both many- and multi-core architectures exploit parallelism features that offer performance gains and faster computing [2].

The demand for high-performance computing has generally been addressed with costly computational systems. However, in view of the popularity of graphics processing units (GPUs) and the adoption of parallel programming methods, a number of research areas can advance significantly without the need for major investment in computational systems. Examples of these areas include: scientific simulation [3], life sciences [4], statistical modeling [4], emerging data-intensive applications [4], electronic design automation [4], ray tracing and rendering [5], computer vision [6], signal processing [4, 6], and medical image processing and analysis [7–9].

The area of medical image processing and analysis has contributed to significant medical advances [6, 9–14] by integrating systems and techniques that support more efficient clinical diagnosis. These systems and techniques are based on images acquired by different imaging modalities such as, endoscopy [15], X-ray [9], microscopy [16, 17], computed tomography (CT) [18, 19], optical coherence tomography (OCT) [20], magnetic resonance (MR) [7, 21], functional magnetic resonance (fMR) [22, 23], magnetic resonance elastography (MRE) [24], positron emission tomography (PET) [25–27], single photon emission computed tomography (SPECT) [28], and 3D ultrasound computer tomography (USCT) [13].

Medical imaging assists physicians in extracting information for the purposes of diagnosing diseases, surgical intervention, treatment and follow-up of diseases, as well as in designing better rehabilitation plans [8, 22, 29, 30]. Such extraction of relevant clinical information is a complex task requiring advanced computational systems able to process and obtain image-based features accurately and consistently within the shortest possible runtime. As a result, a new research area has emerged that combines computational techniques used for medical image processing and analysis [6, 9, 10] and high-performance computing solutions [11–14]. These two components can be briefly described as follows:

- Medical image processing and analysis - Typically, the researchers of this area attempt to find solutions that start by improving the quality of the input images, and then apply operations on the enhanced images in order to identify and extract meaningful clinical information [6, 9, 10]. In this context, the term “medical image processing and analysis” is used throughout the present review.

- High-performance computing - The main goal of this area is to optimize computational methods to achieve greater robustness, effectiveness, efficiency, and faster execution. To accomplish these objectives, parallel computing techniques are usually exploited to use the maximum available performance in the computational architecture adopted [11–14].

The number of researchers combining techniques of medical image processing and analysis and of high-performance computing has increased considerably in recent years; consequently, this article aims to present an updated systematic literature review of this area. The scientific articles selected for this review provide valuable information for researchers in the two fields identified; specifically, the articles address methods, techniques, imaging modalities, metrics of computational performance, and the most frequently used computing architectures. The contributions made by each selected article are therefore set out and the remaining research gaps are identified; this will be of significant value to those who intend to develop, evaluate and compare algorithms used in medical image processing and analysis accelerated by high-performance computing architectures.

The term "performance" is sometimes ambiguous; hence, in this article, "performance" refers to the efficiency of computing systems when executing algorithms, including the factors of throughput, latency, and availability. The methodology employed to select, identify, and validate the articles considered is presented in Sect. 2; the main findings extracted from the articles analyzed are summarized in Sect. 2.1; the contributions found in the selected articles and the gaps identified are presented and discussed in Sect. 3; finally, concluding remarks are presented in Sect. 4.

2 Systematic literature review

This section describes the protocol used to locate, gather, and appraise the state of the art under study. The first issue that was examined was the range of high-performance computing platforms and methods that have been used to speed up techniques of medical image processing and analysis. In addition, the following complementary questions were considered:

1. Which imaging modality was involved?
2. Which task of medical image processing and analysis was addressed?
3. Which human organ or tissue was analyzed?
4. What computational architecture was adopted and/or developed?
5. Which high-performance computing technique was adopted and/or developed?
6. Is the approach adopted and/or developed able to achieve real time?

The criteria defined for the selection of articles are as follows:

1. Domain
   (a) Medical image processing and analysis; and
   (b) High-performance computing.
2. Methods
   (a) Techniques of medical image processing and analysis accelerated by high-performance computing solutions.
3. Measures
   (a) Techniques of medical image processing and analysis; and
   (b) Performance in runtime.

After defining the selection criteria, the next step involved defining the exclusion criteria, which were as follows:

1. Duplicated references; for example, the same article retrieved from the different electronic repositories searched;
2. Less than four pages;
3. No description available on the technique of medical image processing and analysis;
4. No information available on the metric used to assess computing performance;
5. None of the research questions under consideration (numbered 1-5) are addressed.

Before initiating the article-gathering process, the language of the articles, the research domains, and the electronic repositories to be considered were defined. We decided to only review articles written in English, the dominant language used in the scientific domains of computer science and engineering. The repositories selected for searching were: *Scopus* [2] and *Web of Science* [3].

The systematic review was carried out from March 2016 to August 2016, and updated in March 2017. Table 1 presents the search terms used when querying each repository and the total number of articles retrieved.

[2] `http://www.scopus.com` - Science Direct.
[3] `http://apps.webofknowledge.com` - Web of Science Core Collection.

Table 1: Total number of articles retrieved from each electronic repository

Repositories	Queries Performed	No. of Articles
Scopus	TITLE-ABS-KEY ((“medical image” OR “medical imaging”) AND (“high performance computing” OR “parallel programming” OR “parallel computing" OR “real-time processing")) AND (LIMIT-TO (DOCTYPE , “cp”) OR LIMIT-TO (DOCTYPE , “ar”))	421
Web of Science	Filtering using the same queries searched above	2, 158
	Total	**2,579**

The search of the Web of Science repository was defined in order to locate the articles related to each of the following queries: a) “*medical image*” OR “*medical imaging*”, b) “*high performance computing*” OR “*parallel computing*” OR “*parallel programming*” OR “*real-time processing*”. These queries were combined using the AND logical operator in order to mimic the equivalent searches in the other repository. “*image processing*” was not used in the search because it could generalize the results too much; instead, the purpose of using “*medical image*” and “*medical imaging*” was to gather all scientific articles related to techniques of medical image processing and analysis.

After removing the 467 duplicate references, each of the remaining 2, 112 articles were then filtered according to the selection criteria, as shown in Table 2. The selection criteria were applied systematically to the title, keywords, and abstract of the articles in the electronic repositories searched, and this resulted in 594 articles. The content of each abstract was initially analyzed with the aim of identifying evidence of the use of high-performance computing architectures in order to support the acceleration of techniques of medical image processing and analysis.

Table 2: Total articles retrieved, duplicated and remaining after applying each criteria

Repositories	Retrieved	Duplicated	Selection criteria	Exclusion criteria
Scopus	421	17	288	32
Web of Science	2, 158	450	306	55
Total	2, 579	467	594	87

Additionally, each article was classified according to three priority levels:

- *Prio-1*: Articles that are very relevant and suitable for the review such that there was evidence of the (previously defined) article-extraction criteria in the title, abstract, and even keyword fields;

- *Prio-2*: Articles that are less important but still suitable;
- *Prio-3*: Articles that may be relevant to other related research, but are not main sources of knowledge for this review.

The classification priorities of the articles selected from each repository are indicated in Table 3. The values shown in this table indicate the suitability of each repository relative to each classification priority previously enumerated.

Table 3: Relevance of each repository used to retrieve articles related to techniques of medical image processing and analysis combined with high-performance computing solutions

Repository	Prio-1 (%)	Prio-2 (%)	Prio-3 (%)
Scopus	71.64	17.41	10.95
Web of Science	17.82	5.63	76.55

2.1 Review of selected articles

In the evaluation stage, the sections of each article presenting the applicable methodology, results, and conclusions were analyzed, in order to identify important information that answers the research questions (1-5) defined in Sect. 2.

In this review, a total of 594 articles were initially selected; however, 507 articles were then removed in accordance with the exclusion criteria, and the 87 remaining articles were analyzed in depth. The exclusion criteria were defined in such a way as to answer the aforementioned, main research questions. Hence, it was critical to identify in each article: the metric(s) used to evaluate computational performance; the high-performance computing architecture and parallel design involved; and the object(s), i.e., tissue(s) or organ(s), addressed by the techniques(s) of medical image processing and analysis. Therefore, during the in-depth analysis of each article, critical information was collected to answer each specific research question.

Table 4 presents in descending chronological order the most relevant information extracted from the 87 articles analyzed, including the description of the main high-performance computing methods applied to the acceleration of the techniques of medical image processing and analysis. The speedup column presents the computational performance results achieved by the authors in respect of the methods studied. Here, speedup is defined as the ratio of the execution time of serial and parallel implementations when both are applied on the same dataset and running on the same computer.

One conclusion drawn from the articles found is that, in recent years, and especially in the last decade, there has been considerable research into the use of techniques of image processing and analysis accelerated by high-performance computing solutions.

The first step in medical imaging consists of acquiring the data using a suitable imaging device and then reconstructing the related images. After that, a number of techniques of image processing and analysis can be applied, such as image reconstruction, image filtering, image segmentation and image registration.

Research	Impact factor	Imaging modality(ies)	Image task(s)	Object(s) analyzed	Parallel architecture(s)	Parallel programming model	Speedup
Miller and Butler, 1993	2.12	CT, SPECT	reconstruction	brain	Massively Parallel Processor (MPP)	SIMD	64×
Kerr and Bartlett, 1995	0.90	CT, SPECT	reconstruction	cardiac	MPP	SIMD	139×
Higgins and Swift, 1997	0.30	CT	reconstruction	cardiac	MPP	SIMD	5×
Formiconi et al., 1997	0.45	CT, SPECT	reconstruction	brain	MPP	MIMD	135×
Christensen, 1998	1.57	CT	registration	craniofacial	MPP and Cluster	SIMD and MIMD	20×
Daggett and Greenshields, 1998	7.68	MRI	classification	bladder and urethra	Cluster	SPMD	6×
Warfield et al., 1998	5.10	CT, MRI	registration	brain	Cluster	MIMD	15×
Saiviroonporn et al., 1998	2.10	CT, MRI	segmentation	bones, aorta, kidneys, skin, brain	MPP	SIMD	10×* 4
Yip et al., 1999	0.61	MRI	reconstruction	skull	Cluster	MIMD	500×
Rohlfing and Maurer, 2003	17.28	MRI, Microscopy	registration	brain and breast	MPP	MIMD	50×
Wachowiak and Peters, 2004, 2006	1.15, 3.72	MRI	registration	brain and heart	Cluster	MIMD	5×
Tirado-Ramos et al., 2004	1.84	MRI and CT	reconstruction	beast	Cluster	MIMD	3×
Doyley et al., 2004	1.84	MRE	reconstruction	beast	Cluster	MIMD	3×
Salomon et al., 2005	1.66	MRI	registration	brain	Cluster	MIMD	10×
Eidheim et al., 2005	0.91	ultrasound	segmentation	liver	GPU	SIMT	34×*
Crane et al., 2006	0.90	MRI	reconstruction	brain	Cluster	MIMD	3×
Deng et al., 2006	2.90	CT	reconstruction	shepp-Logan phantom	Cluster	MIMD	32×
Dandekar and Shekhar, 2007	4.6	CT, PET	registration	abdominal	FPGA	SIMD	30×
Yeh and Fu, 2007	1.5	fMRI	classification	brain	Cluster	MIMD and SPMD	2×
Kalmoun et al., 2007	2.7	CT	reconstruction	heart	Cluster	MIMD	28×
Kumar et al., 2008	2.22	Microscopy	reconstruction	breast	Cluster	MIMD	2×
Samant et al., 2008	12.77	4DCT	registration	lung	GPU	SIMT	56×
Sehellmann et al., 2008	2.77	PET	reconstruction	lung	GPU	SIMT	7.5×
Melvin et al., 2008	0.66	CT	reconstruction	shepp-Logan phantom	multi-core	SIMD	30×
Kegel et al., Kegel et al., 2009, 2011	3.62, 1.14	PET	reconstruction	rats	multi-core	SPMD	3×
Rehman et al., 2009	5.62	MRI	registration	brain	GPU	SIMT	965×
Rohrer and Gong, 2009	0.37	CT, MRI	registration	abdominal	CBEA	SIMD and MIMD	13×*
Zhuge et al., Zhuge et al., 2011, 2009	2, 3.33	CT, MRI	segmentation	head	GPU	SIMT	18×*
Moyano-Avila et al., 2009	0	X-Ray	reconstruction	vessels	MPP	MIMD	15×
Chung et al., 2010	1.57	microscopy	reconstruction	viruses	GPU	SIMT	16×
Shackleford et al., 2010	14	3D CT	registration	lung	GPU	SIMT	15×*
Shams et al., Shams et al., 2010, 2010	11	CT, MRI, PET	registration	brain	GPU	SIMT	50×*
Gabriel et al., 2010	2.57	FNAC	segmentation	thyroid	Cluster and multi-core	MIMD and SIMD	11×
Lapeer et al., 2010	2.57	CT, MRI	registration	head	GPU	SIMT	10×
Zhu and Cochoff, 2010	1.42	CT, PET	registration	lung	multi-core	SPMD	2-10×
D'Amore et al., 2011	1	MRI	segmentation	skin	multi-core	SIMD	6×
Meng et al., 2011	4.5	CT	reconstruction	lung	cloud computing	MIMD	10×
Schmid et al., 2011	2.66	MRI	segmentation	bones	GPU	SIMT	70×
Schellmann et al., 2011	2.33	PET	reconstruction	mouse	GPU	SIMT	2×
Gao et al., 2011	1.16	MRI	segmentation	brain	GPU	SIMT	1440×*
Lee et al., 2012	7.8	MRI	registration	brain	GPU	SIMT	129×
Adeshina et al., 2012	1.66	MRA	reconstruction	brain	GPU	SIMT	3×

Research	Impact factor	Imaging modality(ies)	Image task(s)	Object(s) analyzed	Parallel architecture(s)	Parallel programming model	Speedup
Murphy et al., 2012	22.4	MRI	reconstruction	torso	GPU and multi-core	SIMT and SIMD	40×
Zinterhof, 2012	0	CT	classification	kidney	GPU	SIMT	120×
Shi et al., 2012	0.40	CT, MRI	segmentation and reconstruction	head, breast, vessels	GPU and multi-core	SIMT and SIMD	40×*
Rodrigues and Bernardes, 2012	2	OCT	filtering	retinal	GPU	SIMT	18×*
Domanski et al., 2013	1.25	CT	reconstruction	brain	GPU and multi-core	SIMT and SIMD	9×
Treibig et al., 2013	5.75	CT, X-ray	reconstruction	rabbit	multi-core	SIMD	6×
Gallea et al., 2013	1.28	CT, MRI	registration	brain	GPU	SIMT	100×
Saran et al., 2014	1.33	MRI	segmentation	breast	GPU and multi-core	SIMT and SIMD	35×
El-Moursy et al., 2014	0.66	3D MRI	segmentation	brain	Cluster	MIMD	2.6×
Balla-Arabé and Gao, 2014	1.33	MRI	segmentation	breast	GPU	SIMT	6×*
Eklund et al., 2014	9	fMRI	registration, segmentation, filtering	brain	GPU	SIMT	195-525×
Barros et al., 2014	0	CT	segmentation	brain	GPU	SIMT	36×
Alvarado et al., 2014	2.33	CT, PET, MRI	segmentation	brain	GPU and multi-core	SIMT and SIMD	8×
Birk et al., 2014	7	USCT	reconstruction	breast	GPU and multi-core	MIMD	25×*
Blas et al., 2014	2	CT	reconstruction	rats	GPU and multi-core	SIMT and SIMD	2×
Mafi and Sirouspour, 2014	3.33	MRI	reconstruction	stomach	GPU	SIMT	28×*
Meng, 2014	1.33	CT	registration	thorax	GPU	SIMT	255×
Wei et al., 2014	0.33	MRI	reconstruction	eye optics	GPU	SIMT	100×
Fan and Xie, 2015	0	CT	reconstruction	shepp-Logan phantom	GPU	SIMT	20×
Serrano et al., 2015	0.5	CT	reconstruction	human body	GPU and Cluster	SIMT and MIMD	22×
Gates et al., 2015	8	CT	segmentation	brain	GPU	SIMT	43.5×
Akgun et al., 2015	1.50	fMRI	segmentation	brain	GPU and multi-core	SIMT and SIMD	157×
Tan et al., 2015	0	microscopy	reconstruction	virus	FPGA, GPU and multi-core	SIMD, SIMT and MIMD	14×
Mahmoudi and Manneback, 2015	1.50	X-ray and MRI	segmentation	vertebra	multi-core and multi-GPU	SIMD and MIMD	98×*
Johnsen et al., 2015	7.50	MRI	registration	breast	GPU	SIMT	5×
Hamdaoui et al., 2015	0	MRI	reconstruction	brain	FPGA	SIMD	37×
Cai et al., 2015	2.50	MRI	registration	lung	GPU and multi-core	SIMT and SIMD	4×
Smistad et al., 2015	0	CT, 3D ultrasound	filtering and segmentation	bone structure, retina blood vessels	GPU and multi-core	SIMD and SIMT	20×
Gulo et al., 2016	3	Ultrasound	filtering	stomach	GPU	SIMT	10×
Nguyena et al., 2016	4.50	MRI	filtering	brain	GPU, Cluster, and multi-core	SIMT, MIMD and SIMD	510×
Koestler et al., 2016	3.50	X-ray	reconstruction	head	GPU	SIMT	1.6×
Hu et al., 2016	1	CT	reconstruction	thorax	GPU	SIMT	202×
Du et al., 2016	0	CT, MRI	registration	brain, lung	GPU	SIMT	17×
Ellingwood et al., 2016	1	CT	registration	lung	GPU	SIMT	112×
Heras et al., 2016	2	MRI,CT	segmentation	brain	GPU	SIMT	6×
Chen et al., 2016	7	Ultrasound	reconstruction	forearm	GPU	SIMT	60×
Aitali et al., 2016	2	MRI	segmentation	skin	GPU	SIMT	52×
Riegler et al., 2016	4	endoscopy	classification	gastrointestinal	multi-core and GPU	SIMD and SIMT	10×
Pang et al., 2016	7	ultrasound	segmentation	breast	GPU	SIMT	16×
Wang et al., Sabne et al., 2016, 2017	4, 2	CT	reconstruction	lungs	GPU	SIMT	4×
Jaros et al., 2017	2	CT	segmentation	heart and liver	GPU	SIMT	44×

Table 5: The Impact Factor column was calculated using the ratio of the number of Google citations of the paper and the number of years since its publication.

2.1.1 Image reconstruction

Image reconstruction is the process used to generate 2D/3D images of an object from the data, i.e., signals, acquired by an imaging device. In the data acquisition stage, the imaging device is responsible for converting the anatomical/physiological information into digital signals. However, digital signals are easily corrupted by noise introduced by the electronic/mechanical components of the imaging device [94]. Dominant physical effects such as resolution, attenuation and scatter, are spatially variant, and in the cases

of attenuation and scatter, may also differ according to the type of object, i.e., tissues, under study. In addition, a number of noise source displacements occur when acquiring MRE images. Lengthy extended movements produce common ambiguity errors, which, for example, result in weak estimates in regions with low signal noise rate. Susceptible effects generate inconsistencies during the estimation stage and result in erroneous estimate displacements. In general, all the image reconstruction approaches demand high computational costs and require large memory capacity, for example, in MRI, SPECT and CT cases, where large datasets are used to reconstruct complex 3D images.

The article of Miller and Butler [18] considers the implementation of the maximum a posteriori (MAP) and maximum likelihood (ML) methods in a system that creates a complete 3D reconstruction from CT images and is accelerated by massively parallel processors. The iterative expectation-maximization (EM) algorithm, which is applied in order to generate ML and MAP estimates for SPECT image acquisitions, is considered highly complex in terms of computation [18]. Their parallel system was implemented on a massively parallel computer (DECmpp-SX 128 $\times$ 128 processor) and designed according to the single instruction, multiple data stream (SIMD) parallel programming model. Although the implementation did not indicate a linear scalability, the speedup achieved was 64x, relative to an optimal programmed implementation to be executed in a reduced instruction set computing (RISC) architecture (64 $\times$ 64 processor). Formiconi et al. [28] also presented a parallel implementation of the EM algorithm; however, their approach was combined with ML estimates and applied in order to reconstruct images from SPECT data. The authors designed their implementation on the basis of a multiple instruction, multiple data stream (MIMD) parallel programming model and used a World Wide Web (WWW) interface. A massively parallel computer, Cray T3D, was used to calculate their computational solution remotely.

Massively parallel computers were adopted by Kerr and Bartlett [31] as described in another article. The authors examined the simulation and rapid training of a very large artificial neural network that reconstructs and compresses SPECT images. In this study, when comparing the performances obtained by CPU- and Parallel-based implementations, a speedup of 139$\times$ was achieved. The authors designed the suggested algorithm on the basis of the SIMD model.

Another research study that developed a parallel computer architecture was presented in the Higgins and Swift [29]'s article. These authors defined a "meta-computer" as a combination of communication devices and a heterogeneous processing architecture. Their goal was implement a new parallel architecture using the parallel computer MasPar in order to manage multiple workstation interactions and process 3D medical images as fast as possible. The parallel architecture used in the experiments included typical

tasks of medical image processing and analysis: image preprocessing, morphological and topological image operations, image segmentation, image manipulation, image measurement and the input and output of images. The approach of the authors resulted in a performance 5× faster than the equivalent algorithm implemented using a sequential fashion programming model.

Doyley et al. [24] proposed in their article a parallel approach to obtain partial volume reconstructions from 3D high-resolution data. The authors combined the finite element method (FEM) and the Newton-Raphson iterative scheme in this approach, which was implemented using Message Passing Interface (MPI) and executed on a PC-cluster. In the experiments, the authors adopted an optimized sequential approach in contrast to a parallel-based one. The parallel version improved the in/out storage disk operations and achieved a linear speedup.

Kumar et al. [17] developed a middleware system based on a PC-cluster architecture, the purpose of which was to support the execution of a set of techniques of image processing and analysis. These techniques were divided into two main stages: preprocessing and analysis. These tasks resulted in preprocessed data that could be queried and analyzed using the techniques of image analysis. The authors combined data and task parallelism models in order to achieve better scalability; moreover, they implemented the tasks of image processing and analysis by changing the number of processors in the PC-cluster; in the experiments performed, a 2× speedup was obtained with the best cluster configuration found.

In the approach of Kegel et al. [26, 27], the Threading Building Blocks (TBB) library and the OpenMP application programming interface were adopted and compared in order to evaluate programming effort, programming style and abstraction, and runtime performance. The authors presented several implementations for systems that support shared- and distributed memory of the list mode ordered subset expectation maximization (LM OSEM) algorithm, resulting in reducing of the processing time spent on reconstruction of PET images. LS OSEM is a computationally intensive block-iterative algorithm for 3D image reconstruction. The authors concluded that the TBB library is much easier to implement than OpenMP, especially when starting a new implementation to exploit parallelism; however, they did not analyze the exact influence of the grain, the block size, or the scheduling strategy for different amounts of input data on the program performance.

The approach presented by Murphy et al. [62] consists of an optimized iterative method, self-consistent parallel imaging (SPIRiT), combined with compressed sensing

for image reconstruction. This approach allows auto-calibrating parallel imaging[5] reconstructions with clinically feasible runtimes. The purpose was to achieve real-time performance via an hybrid implementation using both multi-GPU and multi-core CPUs as parallel execution platforms. Two data parallelism models, SIMD and SIMT, were exploited and optimized through Streaming SIMD Extensions (SSE) and compute unified device architecture (CUDA) instructions, respectively. Parallel GPU and CPU implementation achieved the speedup of 40× when comparing with the runtime of a sequential C++ implementation using high-performance libraries and compiled with full compiler optimization.

Domanski et al. [3] developed a Cluster web services (CWS) framework capable of taking advantage of massively parallel technologies composed of a PC-cluster [6] and GPUs [7]. This framework facilitated communication between the client and server through the Internet in order to balance and distribute the computational load. Although the framework was able to solve a wide range of scientific problems, its main application was the full reconstruction of CT images. The parallel programming languages adopted were Open Computing Language (OpenCL) and MPI, for the GPU architecture and the PC-cluster, respectively.

Treibig et al. [9] presented an approach to the achievement of optimal performance according to the processor specifications and different optimization levels. The authors presented a number of low-level optimizations and algorithms for a back-projection reconstruction strategy from CT data, running on multi-core processors. The implementation was based on SSE and Advanced Vector Extensions (AVX) instructions. The result of this approach was a speedup of up to 6×; however, the authors considered that further studied were needed (a) to improve the implementation performance using distributed memory, (b) to optimize and analyze the AVX kernel update, and also (c) to include the new AVX2 operations collector.

Blas et al. [71] described the performance optimization process of a modular application based on a GPU architecture using the Feldkamp, Davis and Kress (FDK) reconstruction algorithm. However, even though the authors performed most parallelization procedures using the SIMT model, the projection decomposition step was performed using the SIMD model and the Open Multi-Processing (OpenMP) language. The experiments were conducted with different multi-GPU configurations and code optimization levels, and a speedup of up to 2× was achieved relative to the

[5] Parallel imaging is a well-established acceleration technique based on the spatial sensitivity of array receivers [62].

[6] 32 Intel Xeon CPU cores.

[7] 6 NVIDIA cards with Tesla GPU.

implementations discussed in their own literature review. Meng et al. [56] accelerated the FDK algorithm using MapReduce in a cloud computing environment. Map functions were used to filter and back-project subsets of projections, and Reduce function to aggregate those partial back-projections into the whole volume. The findings of this approach were the reconstruction time achieved, whose correlation with the number of nodes employed was roughly linear. Experiments showed a speedup of 10× using 200 nodes for all cases, when compared to the same code executed on a single machine.

Birk et al. [13, 95] adopted multi-GPU and multi-core as a parallel architecture in order to accelerate 3D reconstructions based on ray casting from ultrasound data. Their approach was extended to identify the ideal number of GPUs required to reconstruct high-resolution image volumes, especially when the processing load had substantially greater DRAM capacity than the CPU system. However, the approach was not able to display in real time the high-resolution images at the pre-visualization stage. The experiments took into consideration the implementation of the optimized method for both architectures: multi-core and multi-GPU. The authors emphasized that they combined SIMT and SIMD parallel programming models.

Wei et al. [5] presented a work that used a ray tracing technique to simulate retinal image formations. This approach simulated realistic light refraction through ocular structures in 3D using polygonal meshes and GPU parallel computing.

Chen et al. [88] described a novel imaging system for real clinical applications. The system could provide incremental volume reconstructions and volume rendering; it could also generate high-quality 3D ultrasound strain images in near real-time due to GPU-based implementation. The approach achieved a 60× speedup compared to a CPU-based implementation. However, it could not provide real-time imaging because the time spent on complex data processing and data transfer was excessive.

2.1.2 Image filtering

Rodrigues and Bernardes [20] improved the process of speckle noise reduction for visual analysis of medical images like optical coherence tomography. The authors proposed preserving edges and other relevant features through filter expansion from 3D OCT images of the posterior segment of the human eye for the adaptive complex-diffusion filter. Their implementation was divided into an environment setup stage and four other stages that were called iteratively. CUDA kernels were considered in parallel convolutions, parallel reductions, and element-wise arithmetic operations over the inputs.

Nguyena et al. [83] presented a hybrid parallelization scheme the aim of accelerate the NL-Means filter algorithm. In their approach, the authors divided the input 3D MRI

volume into sub-volumes in order to reduce the search region at the boundary zone. Then the image was divided into superimposed images and the superposition of the search region radius. In the implementation stage, the following parallel technologies were used: MPI, multi-threading on multi-core machines and GPUs. Communication between each cluster node was enabled by using MPI. The main contributions of the authors are an approach that requires different modes of implementation and the possibility of using the MPI technology alone or in conjunction with POSIX Threads (Pthreads) and GPUs. This latter approach reduced the computational time by a factor of approximately 510 when applied to 3D medical data. On the other hand, high memory usage emerged as a drawback of this approach, with up to three times more memory required than with the original method.

Gulo et al. [82] described in their study how to use the high-performance computing CUDA-based architecture as a computational infrastructure to accelerate an algorithm for noise image removal. The parallel GPU-based implementation developed was compared against the corresponding sequential CPU-based implementation in several experiments. The parallelization of the image smoothing method based on a variational model using CUDA architecture reduced the runtime by up to 10 times in comparison with the CPU-based implementation.

2.1.3 Image Segmentation

Image segmentation is one of the most important operation of the image processing and analysis area, being responsible for identifying and delineating objects of interest in input images. In general, tasks of 3D visualization, interpolation, filtering, classification, and even registration depend heavily on the image segmentation results in order to achieve optimum performance [11, 96, 97]. There are several approaches of image segmentation based on, for example, thresholding [33, 67], clustering [8] and deformable models [38].

Daggett and Greenshields [7] designed a parallel algorithm using a PC-cluster to segment MRI images by means of automatic image classification in order to reduce the inter-process communication overhead. This parallel algorithm was based on the virtual shared memory technique, which enables processes to communicate by directly sharing data as though it existed in a global shared memory space. The main idea was to segment anatomical images in order to obtain quantitative anatomical features and geometrically-shaped models of the objects under study.

In the article of Yeh and Fu [22], an approach called parallel adaptive simulated annealing was developed to assist computer-aided measurements for identifying the associated activation regions of the brain through response waveform of functional MR

images. This approach was based on a coarse-grained model performed on a cluster of four PCs; it was designed using the MPI parallel programming language and the single program, multiple data stream (SPMD) data decomposition model. The purpose of this parallelism was to reduce the computational time required by the minimization of the weighted sum of the squared Euclidean distances between each input vector and the prototypes. Additionally, it was able to automatically make clinical diagnoses of schizophrenia and multiple sclerosis.

Gabriel et al. [8] suggested the Gabor filtering system for texture-based image segmentation of thyroid cells. This approach was based on distributed memory and exploited a PC-cluster and the current multi-core CPU architecture. The authors combined several metrics to evaluate the performance of their approach; they then used OpenMP and MPI to compare the speedup, communication overhead, the different memory systems, and the different number of threads used. The multi-core architecture achieved the highest speedups, which were up to $11\times$ faster compared to the PC-cluster. Although the authors presumed that their computational system would be able to make medical diagnoses, their implementation did not have a module for image analysis, or even a tool for the addition of an image set combined with the related diagnosis result.

Zhuge et al. [11, 97] developed a semi-automatic segmentation method based on the fuzzy connected technique, which was implemented using a GPU architecture. Moreover, they designed a robust and efficient parallel version of Dijkstra's algorithm in a SIMD model. This new approach took advantage of the CUDA architecture, especially by supporting atomic read/write operations in the GPU global memory.

Shi et al. [12] proposed an automatic image segmentation method for medical images based on a pulse coupling neural network combined with the 2D Tsallis entropy. Stronger adaptability, high image segmentation precision, and adequate image reconstruction from CT and MR data were the main advantages of this approach. The achievement with this GPU-based approach was the rendering of 3D volume images in real time using ray tracing implemented using a SIMT model.

In the approach by Saran et al. [65], the rigid registration of magnetic resonance venography (MRV) images and magnetic resonance angiography (MRA) images based in mutual information is performed to increase the accuracy of vessels segmentation in MRI images. The unfavorable effects of Rician noise and RF inhomogeneity in the MRI, MRA, and MRV images during vessels segmentation are removed by applying a subtraction schema where the cost function and choice of minimization method are executed simultaneously using multi-core and GPU.

Balla-Arabé and Gao [67] presented a new level set method (LSM) for image segmentation. The authors designed a selective entropy-based energy functional method,

robust against noise, and new selective entropy external forces for the Lattice Boltzmann method (LBM). The LSM and LBM were combined and implemented on GPUs. However, LBM requires significant memory and the approach did not achieve volume image segmentation in real time. Hence, the authors identified a need for future studies to extend their approach to a GPU cluster environment.

Aitali et al. [21] exploited the performance of GPU to accelerate a Bias Field Correction Fuzzy C-Means algorithm used for segmenting MR images. This approach was applied to correct the inhomogeneity intensity and segment the images simultaneously. However, the expensive computation required by the algorithm demanded optimization strategies in order to reduce the runtime; hence, the authors adopted the SIMD architecture to model their approach. The GPU implementation achieved about 52x$\times$ speedup relative to the CPU implementation, and consisted of a novel SIMD architecture for bias field estimation and image segmentation.

Heras et al. [87] used GPU features to accelerate the Fast Two-Cycle method, which is a level set-based segmentation method. In their approach, they aimed to divide the active domain into fixed-size tiles and therefore intensively use shared memory space, resulting in a low latency close to that of the register space. Although the authors did not use real images, they measured the performance of their approach using a set of realistic MRI data volumes produced by an MRI simulator. The volumes produced by this simulator are available to be downloaded at the BrainWeb Simulated Brain Database[8] and they have been broadly used in other published articles. In the experiments, the GPU approach achieved about 6$\times$ speedup relative to the CPU implementation.

2.1.4 Image registration

Image registration is a computational task that establishes a common geometric reference frame across two or more image datasets; it is required, for example, in the comparison or fusion of image data obtained at different times or using different imaging modalities or devices [16, 46]. Intensity-based registration techniques are accurate, efficient, and robust; in addition, they depend on the interpolation scheme, search space, a similarity metric, and an optimization approach [36]. Consequently, these techniques are based on geometric transformations [32], optimization algorithms [36], and measures of similarity [19, 25].

The mutual information-based (MI-based) deformable registration algorithm was considered promising by Dandekar and Shekhar [25], mainly because it was able to correct the misalignment of tissue in CT slice images. The authors demonstrated a

[8] BrainWeb Simulated Brain Database - `http://www.bic.mni.mcgill.ca/brainweb`.

registration accuracy comparable to one achieved by a group of clinical experts [25, 30]. Computationally, MI-based registration is extremely intensive and so requires several thousand of iterations, with the precise number depending on the degree of the initial misalignment, the transformation complexity, the image content, and the optimization algorithm used to maximize the MI function. In order to reduce the runtime on the order of minutes or seconds, and thereby become suitable for clinical routine use, MI-based algorithms have been accelerated in parallel architectures such as clusters [32, 64], GPU [10, 64, 72], multi-core cell broadband engine architecture (CBEA) [47], and field programmable gate array (FPGA) [25].

Christensen [32] developed a 3D linear elastic transformation model using an SGI Challenge parallel computer in order to generate global non-rigid deformations of template image volumes. This approach was optimized to maximize the ratio of computation to the parallelization overhead. In this research, parallel overhead consisted of the runtimes for creating processes, starting and ending parallel regions, and running extra code required for parallelization. The authors performed experiments using implementations optimized for MasPar (SIMD) and Challenge (multiple instruction, multiple data (MIMD)) parallel architectures. The MIMD parallel programming model achieved speeds of up to 20× greater than the SIMD model.

Warfield et al. [30] presented a new registration algorithm that identifies features in image scans which need to be aligned and find the transform that minimizes the mismatch of corresponding tissue labels. This approach was implemented on a parallel platform in order to conform to a clinically acceptable timeframe. The authors adopted a multi-core PC-cluster and the MPI language as the high-performance computational infrastructure to perform the experiments; their approach was designed based on the MIMD-based parallel programming model.

Rohlfing and Maurer [16] solved problems related to the high computational efforts that are commonly incurred when non-rigid image registration techniques are used. The authors took advantage of shared-memory multiprocessor computer architectures as well as data and task partition parallel programming models. Non-rigid image registration techniques demand lengthy execution times because of the input images are usually large and because the adopted transformation model requires substantially more time to compute and evaluate the similarity measure used. The experiments were performed on an SGI Origin 3800 massively parallel computer, and all the results were compared using different degrees of parallelism (2, 16, 32, and 48 threads); the performance achieved showed a reduced linear execution time.

Salomon et al. [38] presented a parallel implementation of a deformable image registration approach based on the multi-resolution technique. In this study, the authors

designed their implementation by applying the MIMD parallel programming model and the OpenMP parallel programming language. However, the SIMD parallel programming model can be considered most suitable when a large number of processors are used. This parallel approach achieved a speedup of up to 10× when applied to the registration of 3D MR images.

Wachowiak and Peters [36] developed two methods - DIviding RECTangles (DIRECT) and Multi-Directional Search (MDS) - that were used to optimize a similarity metric, which is an essential component of intensity-based medical image registration algorithms. The DIRECT method was employed as a global technique for linearly bounded problems and was followed by local refinements attained with the MDS method. This approach was implemented and optimized for execution in shared memory systems. With the use of 8 or 12 CPUs on a PC-cluster, the results demonstrated efficiency gains, yielding a speedup of up to 5×.

Rehman et al. [46] employed GPU architecture to achieve high performance using the multi-resolution approach that is typically applied in non-rigid 3D image registration. In this article, the authors developed a parallel approach of non-rigid registration by regarding it as an optimal mass transport problem. The experiments showed a speedup improvement in the parallel architecture of up to 965× relative to the CPU-based implementation.

Rohrer and Gong [47] and Shams et al. [10] enabled different high-performance computing architectures to achieve real-time image registration. Rohrer and Gong [47] combined mutual information and multi-resolution techniques, and implemented them on a heterogeneous multi-core architecture called CBEA. The implementation of this approach on a GPU architecture Shams et al. [10, 52] made an innovative contribution to the computing of MI by computing joint histograms. On the basis of this approach, the registration of 3D CT, PET and MR images was achieved in real time.

Assuming relatively small non-linear displacements and deformations in the registration of CT and MRI data related to the head, Lapeer et al. [53] presented a point-based registration method. This new method was developed in order to speed up a nonlinear multimodal registration algorithm on a GPU architecture. The approach integrated the radial basis function (RBF) as a smooth function and sought to mimic the interacting deformation of biological tissues. The performance tests demonstrated that the GPU-based implementation yielded a run-time 10× faster than that of the CPU-based implementation.

Zhu and Cochoff [54] demonstrated how to use parallel programming patterns aiming to obtain better performance in applications relating to image visualization, registration, and fusion. The parallel programming pattern used depends on the architecture adopted.

Thus, it can involve data parallelism, task parallelism, coordination based on events, data sharing, asynchronous calls, and fork/join. Using multi-core and symmetric multiprocessor (SMP) architectures, the speed was up to 10× faster relative to a CPU architecture. In addition, the parallel implementation confirmed the presence of the important features of portability and flexibility.

Mafi and Sirouspour [14] developed a GPU-based computational platform for real-time analysis of soft object deformation. This GPU-based computing scheme solved a large system of linear equations and updates the nonlinear FEM matrices in real time. However, this approach can be extended to even further optimize all computations related to single- and double-precision operations. In addition, it can enable multiple GPU-based computing, deformation analysis with multiple contact points, and auto-adaptive mesh refinement in order to improve analysis accuracy.

Ellingwood et al. [19] presented a novel computation- and memory-efficient Diffeomorphic Multi-Level B-Spline Transform Composite method on GPU for the performance of non-rigid mass-preserving registration of CT volumetric images. The authors adopted the sum of squared tissue volume difference (SSTVD) as the similarity criterion to preserve the lung tissue mass; hence, SSTVD was used for computing the tissue volume. A cubic B-Spline-based free-form deformation (FFD) transformation model was employed for capturing the non-rigid deformation of objects such as human lungs. The experiments used lung CT images, which indicated a speedup of 112 times relative to the single-threaded CPU version, and of 11 times compared to the 12-threaded version when considering the average time per iteration using the GPU implementation. The authors compared the following types of algorithms: single-threaded CPU-based, multi-threaded GPU-based, and GPU-based.

3 Discussion

The deployment of high-performance computing techniques has greatly contributed to reducing the processing time of techniques used for medical image processing and analysis, making them suitable for routine clinical use. Briefly, these techniques were used in order to exploit all the computational power commonly available in modern high-computing architectures such as multi-core, GPU and PC-cluster.

Following the recent advances in GPU [5, 10, 11, 14, 20, 46, 52, 53, 72, 97], multi-core [3, 8, 9, 12, 13, 19, 23, 54, 62, 71, 95], and FPGA [25, 76, 98–100] architectures, researchers have confirmed a trend towards lower computational costs without any consequential reduction in terms of the accuracy of the techniques of image

processing and analysis. Hence, Domanski et al. [3], Shi et al. [12], Birk et al. [13], Murphy et al. [62], Saran et al. [65], Alvarado et al. [70], Serrano et al. [74], Birk et al. [95] designed their models using parallel programming in GPU and multi-core; on the other hand, Blas et al. [71], Tan et al. [76], Mahmoudi and Manneback [77], Cai et al. [80], Riegler et al. [89], Nguyen et al. [101] have demonstrated an approach which is more focused on load-balancing techniques, multi-GPU, GPU, and multi-core architectures. Therefore, there is an increasing number of methodologies that achieve high performance levels and that combine parallel programming methods and high-performance computing architectures; furthermore, the run-time and energy consumption required by these methodologies are decreasing considerably.

The articles evaluated in this review provide an overview on techniques of medical image processing and analysis accelerated by high-performance computing solutions. Figure 1 shows that the majority of the selected articles were published in the last decade and the last five years have seen remarkable progress thanks to multi-core processors and GPU architecture [6]. It is important to highlight that this review covers papers published up to March 2017.

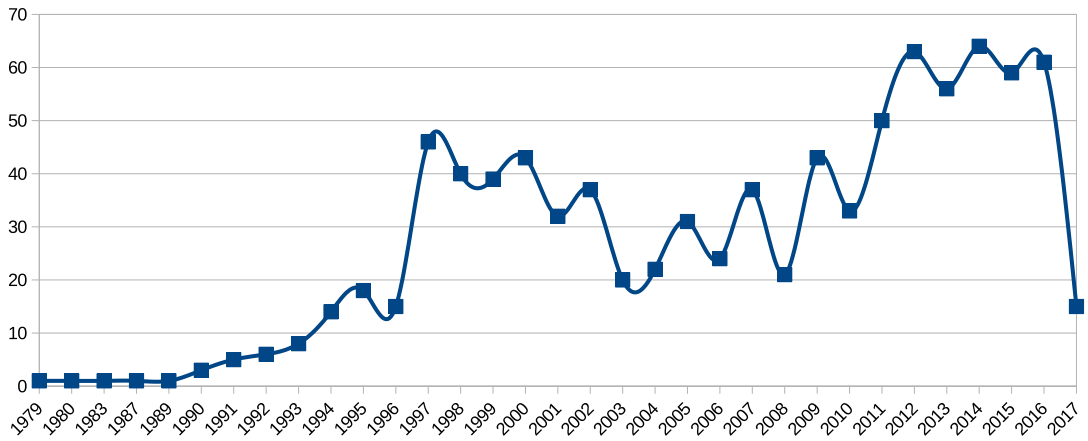


Figure 1: Distribution of selected articles related to techniques of medical image processing and analysis accelerated by high-performance computing solutions published in recent years

Although the articles listed in Table 4 report on highly positive speedup findings, it is important to analyze these results carefully. The majority of the selected articles indicated speedup as the main metric used to evaluate the performance gain. Almost half of the articles compared sequential and parallel implementations, as can be seen in Gabriel et al. [8], Shams et al. [10], Shi et al. [12], Birk et al. [13], Mafi and Sirouspour [14], Rohlfing and Maurer [16], Yeh and Fu [22], Dandekar and Shekhar [25], Rehman et al. [46], Rohrer and Gong [47], Zhuge et al. [48], Shams et al. [52], Lapeer et al.

[53], Zhu and Cochoff [54], Murphy et al. [62], Blas et al. [71], Meng [72], Birk et al. [95]. One of the greatest challenges in this sort of comparison is to describe how well sequential implementation was optimized, and more particularly: (1) whether the SSE instruction set was used; (2) whether the code was compiled in 32 or 64 bits; and (3) whether 32- or 64-bit floating point operations were used. This sort of optimization is critical when comparing implementations that use multi-core, GPU, or cluster architectures. Usually, it is necessary to rewrite code in order to improve application performance and so exploit the benefit of parallelization. As a result, it is good practice to divide an application into smaller tasks that can be executed in parallel [102]. However, during task deconstruction, the communication process and the general coordination of processing jobs among the processors used need to be taken into account.

When adopting a parallel programming design, two main features must be taken into account: (1) the parallel architecture and (2) the type of processor communication [103]. The high computational costs of data access and task performance are dependent on the computational resources available to the computing system. Hence, parallel design should make use of data decomposition and allocate available memory efficiently.

Most of the analyzed articles focused on the parallelizing of techniques of medical image reconstruction and registration. PC-clusters are the parallel infrastructure most often adopted by researchers [7, 17, 22, 24, 30, 33, 38], FPGA [25, 76, 98–100], in addition to the most recent GPU-based technologies [5, 10, 11, 14, 46, 52, 53, 67, 72, 97] and multi-core [3, 8, 9, 12, 13, 19, 23, 54, 62, 71, 95] architectures. Moreover, it is clear that the research topic discussed in this review is recent and promising, as confirmed by the remarkable increase in the number of related scientific articles published in the last decade. In summary, the reviewed articles demonstrated a reduction in the run-time, including in real time, which is ideal for routine medical applications. However, just a few of the selected articles focused on speeding up techniques of medical image segmentation, which suggests a potential topic for further research.

This article presents a concise and up-to-date review of techniques of medical image processing and analysis that have been implemented based on high-performance computing solutions. As a result, related researchers can identify: (a) the GPUs as computing systems, (b) the SIMD as the main parallel programming model, that have been most widely used to deal with the typical demands of techniques of medical image processing and analysis. The most used computing systems are presented in Fig. 2. In particular, this review also reveals that data-parallel computations with high arithmetic intensity are well-suited to SIMD parallelization; then, it is well suited for the computation on GPUs. This is because the execution model of GPUs is based on SIMD parallel programming model, which allows multiple processing elements to perform the

same operation on multiple data, concurrently.

The greatest programming efforts found in the selected articles are: (a) the learning curve required for programming parallel implementations, (b) obtaining a complete understanding of the advanced concepts related to memory hierarchy, (c) and the design of the shortest-possible, optimal data paths.

Usually, modifying the design of a sequential algorithm in order to make it parallel requires changing the programming model, the programming language, and the memory access strategy. Successful implementation of these changes will also achieve maximum performance and a higher optimization level due to lower throughput across different memory types.

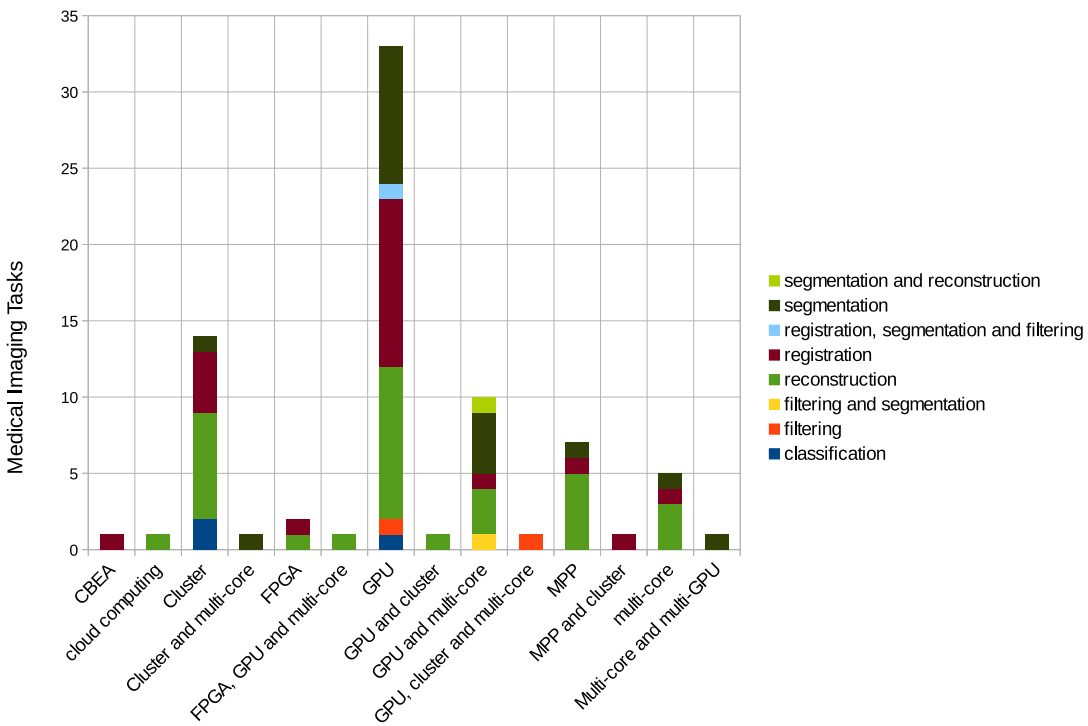


Figure 2: Main parallel programming models applied to accelerate tasks of medical image processing and analysis

4 Conclusion

In this article, the main research articles relating to the combination of techniques of medical image processing and analysis with different high-performance computing solutions have been reviewed. The selected articles describe the use of high-performance

computing systems, including multi-core, GPU, FPGA and PC-cluster, and their capacity to support tasks of medical image processing and analysis.

This article reviewed a set of articles related to complex techniques of medical image processing and analysis, and experiments performed using high-performance computing systems. By combining parallel computer solutions with algorithms of medical image processing and analysis, the scientific community is able to make significant advances in the field of medicine, especially by reducing the required runtime; this in turn enables solutions to be implemented in routine clinical scenarios. Moreover, this article will be useful in developing new research that evaluates and compares different algorithms of medical image processing and analysis supported by high-performance computing solutions.

GPUs are considered to be extremely fast processors, especially when used in computational systems like multi-GPU. On the other hand, the use of multiple GPUs has presented additional challenges; for instance, regarding the efficient management of reading and/or writing data on the data store system, time-consuming data transfers between the CPU and GPU, and load-balancing. The main issue in shared memory systems is that data must be protected against simultaneous access so that errors and data inconsistency can be avoided; additionally, the number of parallel tasks must be at least the same number of processing units (cores), and each task must have enough memory for its computing requirements.

5 Acknowledgments

The first author would like to thank the Universidade do Estado de Mato Grosso (UNEMAT), in Brazil, and the National Scientific and Technological Development Council ("Conselho Nacional de Desenvolvimento Científico e Tecnológico" - CNPq), process 234306/2014-9, grant with reference #2010/15691-0, for the support given. The authors gratefully acknowledge the funding received from Project NORTE-01-0145-FEDER-000022 - SciTech - Science and Technology for Competitive and Sustainable Industries, co-financed by "Programa Operacional Regional do Norte" (NORTE2020), through "Fundo Europeu de Desenvolvimento Regional" (FEDER).

References

[1] D. Kirk and W.-M. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, 2010. ISBN 978-0-12-381472-2.

[2] A. Vadja. *Programming Many-Core Chips*. Springer, 2011. ISBN 978-1-4419-9738-8. doi: 10.1007/978-1-4419-9739-5.

[3] L. Domanski, T. Bednarz, T. Gureyev, L. Murray, B. Huang, Y. Nesterets, D. Thompson, E. Jones, C. Cavanagh, D. Wang, P. Vallotton, C. Sun, A. Khassapov, A. Stevenson, S. Mayo, M. Morell, A. George, and J. Taylor. Applications of heterogeneous computing in computational and simulation science. *International Journal of Computational Science and Engineering*, 8(3):240–252, 2013. doi: 10.1504/IJCSE.2013.055356.

[4] W. mei W. Hwu, editor. *GPU Computing GEMS - Emerald Edition*. Morgan Kaufmann, 2012. ISBN 978-0-12-384988-5.

[5] Q. Wei, S. Patkar, and D. K. Pai. Fast ray-tracing of human eye optics on graphics processing units. *Computer Methods and Programs in Biomedicine*, 114 (3):302–314, MAY 2014. ISSN 0169-2607. doi: 10.1016/j.cmpb.2014.02.003.

[6] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte. Medical image processing on the GPU - past, present and future. *Medical Image Analysis*, 17(8):1073–1094, DEC 2013. ISSN 1361-8415. doi: 10.1016/j.media.2013.05.008.

[7] T. Daggett and I. Greenshields. Parallelization of classification algorithms for medical imaging on a cluster computing system. In *11TH IEEE Symposium on Computer-Based Medical Systems, Proceedings*, pages 305–310. IEEE Comp Soc; IEEE Comp Soc Tech Comm Computat Med; IEEE S Plains Sect-Reg V; Int Soc Optical Engn (SPIE); TX Tech Univ Health Sci Ctr, Dept Radiol, 1998. ISBN 0-8186-8563-8. doi: 10.1109/CBMS.1998.701384.

[8] E. Gabriel, V. Venkatesan, and S. Shah. Towards high performance cell segmentation in multispectral fine needle aspiration cytology of thyroid lesions. *Computer Methods and Programs in Biomedicine*, 98(3):231–240, 2010. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2009.07.008.

[9] J. Treibig, G. Hager, H. Hofmann, J. Hornegger, and G. Wellein. Pushing the limits for medical image reconstruction on recent standard multicore processors. *International Journal of High Performance Computing Applications*, 27(2): 162–177, 2013. doi: 10.1177/1094342012442424.

[10] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. A survey of medical image registration on multicore and the GPU. *IEEE Signal Processing Magazine*, 27(2): 50–60, 2010. ISSN 1053-5888. doi: 10.1109/MSP.2009.935387.

[11] Y. Zhuge, Y. Cao, J. K. Udupa, and R. W. Miller. Parallel fuzzy connected image segmentation on GPU. *Medical Physics*, 38(7):4365–4371, JUL 2011. ISSN 0094-2405. doi: 10.1118/1.3599725.

[12] W. Shi, Y. Li, Y. Miao, and Y. Hu. Research on the key technology of image guided surgery. *Przeglad Elektrotechniczny*, 88(3B):29–33, 2012. ISSN 0033-2097.

[13] M. Birk, R. Dapp, N. Ruiter, and J. Becker. GPU-based iterative transmission reconstruction in 3D ultrasound computer tomography. *Journal of Parallel and Distributed Computing*, 74(1):1730–1743, 2014. ISSN 0743-7315. doi: http://dx.doi.org/10.1016/j.jpdc.2013.09.007.

[14] R. Mafi and S. Sirouspour. GPU-based acceleration of computations in nonlinear finite element deformation analysis. *International Journal for Numerical Methods in Biomedical Engineering*, 30(3):365–381, 2014. doi: 10.1002/cnm.2607.

[15] R. Melo, G. Falcao, and J. Barreto. Real-time HD image distortion correction in heterogeneous parallel computing systems using efficient memory access patterns. *Journal of Real-Time Image Processing*, 11(1):83–91, 2016. doi: 10.1007/s11554-012-0304-3.

[16] T. Rohlfing and J. Maurer, C.R. Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees. *IEEE Transactions on Information Technology in Biomedicine*, 7(1):16–25, 2003. ISSN 1089-7771. doi: 10.1109/TITB.2003.808506.

[17] V. Kumar, B. Rutt, T. Kurc, U. Catalyurek, T. Pan, S. Chow, S. Lamont, M. Martone, and J. Saltz. Large-scale biomedical image analysis in Grid environments. *IEEE Transactions on Information Technology in Biomedicine*, 12 (2):154–161, 2008. ISSN 1089-7771. doi: 10.1109/TITB.2007.908466.

[18] M. Miller and C. Butler. 3D maximum a posteriori estimation for single photon emission computed tomography on massively-parallel computers. *IEEE Transactions on Medical Imaging*, 12(3):560–565, Sep 1993. ISSN 0278-0062. doi: 10.1109/42.241884.

[19] N. D. Ellingwood, Y. Yin, M. Smith, and C.-L. Lin. Efficient methods for implementation of multi-level nonrigid mass-preserving image registration on GPUs and multi-threaded CPUs. *Computer Methods and Programs in Biomedicine*, 127:290 – 300, 2016. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2015.12.018.

[20] P. Rodrigues and R. Bernardes. 3-D adaptive nonlinear complex-diffusion despeckling filter. *IEEE Transactions on Medical Imaging*, 31(12):2205–2212, DEC 2012. ISSN 0278-0062. doi: 10.1109/TMI.2012.2211609.

[21] N. Aitali, B. Cherradi, A. E. Abbassi, O. Bouattane, and M. Youssfi. Parallel implementation of bias field correction fuzzy c-means algorithm for image segmentation. *International Journal of Advanced Computer Science and Applications*, 7(3):375–383, 2016. ISSN 2158-107X.

[22] J.-Y. Yeh and J. Fu. Parallel adaptive simulated annealing for computer-aided measurement in functional MRI analysis. *Expert Systems with Applications*, 33 (3):706–715, 2007. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2006.06.018.

[23] D. Akgun, U. Sakoglu, J. Esquivel, B. Adinoff, and M. Mete. GPU accelerated dynamic functional connectivity analysis for functional MRI data. *Computerized Medical Imaging and Graphics*, 43:53 – 63, 2015. ISSN 0895-6111. doi: http://dx.doi.org/10.1016/j.compmedimag.2015.02.009.

[24] M. Doyley, E. Van Houten, J. Weaver, S. Poplack, L. Duncan, F. Kennedy, and K. Paulsen. Shear modulus estimation using parallelized partial volumetric reconstruction. *IEEE Transactions on Medical Imaging*, 23(11):1404–1416, 2004. ISSN 0278-0062. doi: 10.1109/TMI.2004.834624.

[25] O. Dandekar and R. Shekhar. FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions. *IEEE Transactions on Biomedical Circuits and Systems*, 1(2):116–127, 2007. ISSN 1932-4545. doi: 10.1109/TBCAS.2007.909023.

[26] P. Kegel, M. Schellmann, and S. Gorlatch. Using OpenMP vs. threading building blocks for medical imaging on multi-cores. 5704 LNCS:654–665, 2009. doi: 10.1007/978-3-642-03869-3_62.

[27] P. Kegel, M. Schellmann, and S. Gorlatch. Comparing programming models for medical imaging on multi-core systems. *Concurrency and Computation-Practice & Experience*, 23(10):1051–1065, JUL 2011. ISSN 1532-0626. doi: 10.1002/cpe.1671.

[28] A. Formiconi, A. Passeri, M. Guelfi, M. Masoni, A. Pupi, U. Meldolesi, P. Malfetti, L. Calori, and A. Guidazzoli. World wide web interface for advanced spect

reconstruction algorithms implemented on a remote massively parallel computer. *International Journal of Medical Informatics*, 47(1-2):125–138, 1997. doi: 10.1016/S1386-5056(97)00089-0.

[29] W. Higgins and R. Swift. Distributed system for processing 3D medical images. *Computers in Biology and Medicine*, 27(2):97–115, MAR 1997. ISSN 0010-4825. doi: 10.1016/S0010-4825(96)00042-X.

[30] S. Warfield, F. Jolesz, and R. Kikinis. A high performance computing approach to the registration of medical imaging data. *Parallel Computing*, 24(9-10): 1345–1368, 1998.

[31] J. P. Kerr and E. B. Bartlett. Medical image-processing utilizing neural networks trained on a massively-parallel computer. *Computers in Biology and Medicine*, 25 (4):393–403, 1995. ISSN 0010-4825. doi: 10.1016/0010-4825(95)00017-X.

[32] G. E. Christensen. MIMD vs. SIMD parallel processing: A case study in 3D medical image registration. *Parallel Computing*, 24:1369–1383, 1998. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(98)00062-3.

[33] P. Saiviroonporn, A. Robatino, J. Zahajszky, R. Kikinis, and F. Jolesz. Real-time interactive three-dimensional segmentation. *Academic Radiology*, 5(1):49–56, JAN 1998. ISSN 1076-6332. doi: 10.1016/S1076-6332(98)80011-1.

[34] H. Yip, I. Ahmad, and T. Pong. An efficient parallel algorithm for computing the gaussian convolution of multi-dimensional image data. *Journal of Supercomputing*, 14(3):233–255, NOV-DEC 1999. ISSN 0920-8542. doi: 10.1023/A:1008137531862.

[35] M. P. Wachowiak and T. M. Peters. *Parallel optimization approaches for medical image registration*, volume 3216, pages 781–788. Springer, 2004. ISBN 0302-9743 3-540-22976-0.

[36] M. Wachowiak and T. Peters. High-performance medical image registration using new optimization techniques. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):344–353, 2006. doi: 10.1109/TITB.2006.864476.

[37] A. Tirado-Ramos, P. Sloot, A. Hoekstra, and M. Bubak. An integrative approach to high-performance biomedical problem solving environments on the grid. *Parallel Computing*, 30(9-10):1037–1055, SEP-OCT 2004. ISSN 0167-8191. doi: 10.1016/j.parco.2004.07.010.

[38] M. Salomon, F. Heitz, G.-R. Perrin, and J.-P. Armspach. A massively parallel approach to deformable matching of 3D medical images via stochastic differential equations. *Parallel Computing*, 31(1):45–71, 2005. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2004.12.003.

[39] O. Eidheim, J. Skjermo, and L. Aurdal. Real-time analysis of ultrasound images using GPU. In H. Lemke, K. Inamura, K. Doi, M. Vannier, and A. Farman, editors, *CARS 2005: Computer Assisted Radiology and Surgery*, volume 1281 of *International Congress Series*, pages 284–289, 2005. ISBN 0-444-51872-X. doi: 10.1016/j.ics.2005.03.187.

[40] J. Crane, F. Crawford, and S. Nelson. Grid enabled magnetic resonance scanners for near real-time medical image processing. *Journal of Parallel and Distributed Computing*, 66(12):1524–1533, 2006. doi: 10.1016/j.jpdc.2006.03.009.

[41] J. Deng, H. Yu, J. Ni, T. He, S. Zhao, L. Wang, and G. Wang. A parallel implementation of the Katsevich algorithm for 3-D CT image reconstruction. *Journal of Supercomputing*, 38(1):35–47, 2006. doi: 10.1007/s11227-006-6675-0.

[42] E. M. Kalmoun, H. Kostler, and U. Rude. 3D optical flow computation using a parallel variational multigrid scheme with application to cardiac C-arm CT motion. *Image and Vision Computing*, 25(9):1482–1494, SEP 1 2007. ISSN 0262-8856. doi: 10.1016/j.imavis.2006.12.017.

[43] S. Samant, J. Xia, P. Muyan-Oelik, and J. Owens. High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy. *Medical Physics*, 35(8):3546–3553, 2008. doi: 10.1118/1.2948318.

[44] M. Sehellmann, J. Vörding, S. Gorlatch, and D. Meiländer. Cost-effective medical image reconstruction: From clusters to graphics processing units. pages 283–291, 2008. doi: 10.1145/1366230.1366278.

[45] C. Melvin, M. Xu, and P. Thulasiraman. HPC for iterative image reconstruction in CT. volume 273, pages 61–68, 2008. doi: 10.1145/1370256.1370265.

[46] T. Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum. 3D nonrigid registration via optimal mass transport on the GPU. *Medical Image Analysis*, 13 (6):931–940, 2009. ISSN 1361-8415. doi: http://dx.doi.org/10.1016/j.media.2008.10.008.

[47] J. Rohrer and L. Gong. Accelerating 3D nonrigid registration using the cell broadband engine processor. *IBM Journal of Research and Development*, 53(5), 2009.

[48] Y. Zhuge, Y. Cao, and R. W. Miller. GPU accelerated fuzzy connected image segmentation by using CUDA. *IEEE Engineering in Medicine and Biology Society*, pages 6341–4, 2009. ISSN 1557-170X. doi: 10.1109/IEMBS.2009.5333158.

[49] E. Moyano-Avila, L. Orozco-Barbosa, and F. J. Quiles. Parallel algorithms based on the temporal-window method for non-alternating 3D-WT over angiographies using a multicomputer. *Journal of Signal Processing Systems for Signal Image and Video Technology*, 55(1-3):267–279, APR 2009. ISSN 1939-8018. doi: 10.1007/s11265-008-0188-4.

[50] J. Chung, P. Sternberg, and C. Yang. High-performance three-dimensional image reconstruction for molecular structure determination. *International Journal of High Performance Computing Applications*, 24(2):117–135, 2010. doi: 10.1177/1094342009106293.

[51] J. A. Shackleford, N. Kandasamy, and G. C. Sharp. On developing b-spline registration algorithms for multi-core processors. *Physics in Medicine and Biology*, 55(21):6329–6351, NOV 7 2010. ISSN 0031-9155. doi: 10.1088/0031-9155/55/21/001.

[52] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. *Computer Methods and Programs in Biomedicine*, 99(2):133–146, AUG 2010. ISSN 0169-2607. doi: 10.1016/j.cmpb.2009.11.004.

[53] R. J. Lapeer, S. K. Shah, and R. S. Rowland. An optimised radial basis function algorithm for fast non-rigid registration of medical images. *Computers in Biology and Medicine*, 40(1):1–7, JAN 2010. ISSN 0010-4825. doi: 10.1016/j.compbiomed.2009.10.002.

[54] Y.-M. Zhu and S. Cochoff. Medical image viewing on multicore platforms using parallel computing patterns. *IT Professional*, 12(2):33–41, 2010. doi: 10.1109/MITP.2010.62.

[55] L. D'Amore, D. Casaburi, L. Marcellino, and A. Murli. Numerical solution of diffusion models in biomedical imaging on multicore processors. *Journal of*

Biomedical Imaging, 2011:9:9–9:9, jan 2011. ISSN 1687-4188. doi: 10.1155/2011/680765.

[56] B. Meng, G. Pratx, and L. Xing. Ultrafast and scalable cone-beam CT reconstruction using MapReduce in a cloud computing environment. *Medical Physics*, 38(12):6603–6609, DEC 2011. ISSN 0094-2405. doi: 10.1118/1.3660200.

[57] J. Schmid, J. A. I. Guitian, E. Gobbetti, and N. Magnenat-Thalmann. A GPU framework for parallel segmentation of volumetric images using discrete deformable models. *Visual Computer*, 27(2, SI):85–95, FEB 2011. ISSN 0178-2789. doi: 10.1007/s00371-010-0532-0.

[58] M. Schellmann, S. Gorlatch, D. Meilaender, T. Koesters, K. Schaefers, F. Wuebbeling, and M. Burger. Parallel medical image reconstruction: from graphics processing units (GPU) to grids. *Journal of Supercomputing*, 57(2, SI):151–160, AUG 2011. ISSN 0920-8542. doi: 10.1007/s11227-010-0397-z.

[59] Y. Gao, J. Yang, X. Xu, and F. Shi. Efficient cellular automaton segmentation supervised by pyramid on medical volumetric data and real time implementation with graphics processing unit. *Expert Systems with Applications*, 38(6):6866–6871, JUN 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2010.12.049.

[60] D. Lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky, and A. W. Toga. CUDA optimization strategies for compute- and memory-bound neuroimaging algorithms. *Computer Methods and Programs in Biomedicine*, 106(3):175–187, JUN 2012. ISSN 0169-2607. doi: 10.1016/j.cmpb.2010.10.013.

[61] A. M. Adeshina, R. Hashim, N. E. A. Khalid, and S. Z. Z. Abidin. Locating abnormalities in brain blood vessels using parallel computing architecture. *Interdisciplinary Sciences-Computational Life Sciences*, 4(3):161–172, SEP 2012. ISSN 1913-2751. doi: 10.1007/s12539-012-0132-y.

[62] M. Murphy, M. Alley, J. Demmel, K. Keutzer, S. Vasanawala, and M. Lustig. Fast $l1$ -SPIRiT compressed sensing parallel imaging MRI: Scalable parallel implementation and clinically feasible runtime. *IEEE Transactions on Medical Imaging*, 31(6):1250–1262, 2012. ISSN 0278-0062. doi: 10.1109/TMI.2012.2188039.

[63] P. Zinterhof. High-throughput-screening of medical image data on heterogeneous clusters. 7116 LNCS:368–377, 2012. doi: 10.1007/978-3-642-29843-1_42.

[64] R. Gallea, E. Ardizzone, R. Pirrone, and O. Gambino. Three-dimensional fuzzy kernel regression framework for registration of medical volume data. *Pattern Recognition*, 46(11):3000–3016, NOV 2013. ISSN 0031-3203. doi: 10.1016/j.patcog.2013.03.025.

[65] A. N. Saran, F. Nar, and M. Saran. Vessel segmentation in MRI using a variational image subtraction approach. *Journal of Electrical Engineering and Computer Sciences*, 22(2):499–516, 2014. ISSN 1300-0632. doi: 10.3906/elk-1206-18.

[66] A. A. El-Moursy, H. ElAzhary, and A. Younis. High-accuracy hierarchical parallel technique for hidden markov model-based 3D magnetic resonance image brain segmentation. *Concurrency and Computation-Practice & Experience*, 26(1): 194–216, JAN 2014. ISSN 1532-0626. doi: 10.1002/cpe.2959.

[67] S. Balla-Arabé and X. Gao. Geometric active curve for selective entropy optimization. *Neurocomputing*, 139:65–76, 2014. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2013.09.058.

[68] A. Eklund, P. Dufort, M. Villani, and S. LaConte. BROCCOLI: Software for fast fMRI analysis on many-core CPUs and GPUs. *Frontiers in Neuroinformatics*, 8, MAR 14 2014. ISSN 1662-5196. doi: 10.3389/fninf.2014.00024.

[69] R. Barros, S. Van Geldermalsen, A. Boers, A. Belloum, H. Marquering, and S. Olabarriaga. Heterogeneous platform programming for high performance medical imaging processing. 8374 LNCS:301–310, 2014. doi: 10.1007/978-3-642-54420-0_30.

[70] R. Alvarado, J. J. Tapia, and J. C. Rolon. Medical image segmentation with deformable models on graphics processing units. *Journal of Supercomputing*, 68 (1):339–364, APR 2014. ISSN 0920-8542. doi: 10.1007/s11227-013-1042-4.

[71] J. G. Blas, M. Abella, F. Isaila, J. Carretero, and M. Desco. Surfing the optimization space of a multiple-GPU parallel implementation of a X-ray tomography reconstruction algorithm. *Journal of Systems and Software*, 95: 166–175, 2014. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2014.03.083.

[72] L. Meng. Acceleration method of 3d medical images registration based on compute unified device architecture. *Biomedical Materials and Engineering*, 24 (1):1109–1116, 2014. doi: 10.3233/BME-130910.

[73] Z. Fan and Y. Xie. A block-wise approximate parallel implementation for ART algorithm on CUDA-enabled GPU. *Biomedical Materials and Engineering*, 26(1): S1027–S1035, 2015. ISSN 0959-2989. doi: 10.3233/BME-151398.

[74] E. Serrano, J. Blas, and J. Carretero. A comparative study of an X-ray tomography reconstruction algorithm in accelerated and cloud computing systems. *Concurrency Computation*, 27(18):5538–5556, 2015. doi: 10.1002/cpe.3599.

[75] M. Gates, M. T. Heath, and J. Lambros. High-performance hybrid CPU and GPU parallel algorithm for digital volume correlation. *International Journal of High Performance Computing Applications*, 29(1, SI):92–106, SPR 2015. ISSN 1094-3420. doi: 10.1177/1094342013518807.

[76] G. Tan, C. Zhang, W. Wang, and P. Zhang. SuperDragon: A heterogeneous parallel system for accelerating 3D reconstruction of cryo-electron microscopy images. *ACM Transactions on Reconfigurable Technology and Systems*, 8(4), OCT 2015. ISSN 1936-7406. doi: 10.1145/2740966.

[77] S. Mahmoudi and P. Manneback. Multi-CPU/multi-GPU based framework for multimedia processing. *IFIP Advances in Information and Communication Technology*, 456:54–65, 2015. doi: 10.1007/978-3-319-19578-0_5.

[78] S. F. Johnsen, Z. A. Taylor, M. J. Clarkson, J. Hipwell, M. Modat, B. Eiben, L. Han, Y. Hu, T. Mertzanidou, D. J. Hawkes, and S. Ourselin. NiftySim: A GPU-based nonlinear finite element package for simulation of soft tissue biomechanics. *International Journal of Computer Assisted Radiology and Surgery*, 10(7):1077–1095, JUL 2015. ISSN 1861-6410. doi: 10.1007/s11548-014-1118-5.

[79] F. Hamdaoui, A. Sakly, and A. Mtibaa. Real-time synchronous hardware architecture for mri images segmentation based on pso. In D. Mehdi, A. Aitouch, and M. Chaabane, editors, *2015 4TH INTERNATIONAL CONFERENCE ON SYSTEMS AND CONTROL (ICSC)*, pages 498–503. Natl Sch Engn Sfax; Univ Sfax; Soc Sci Dev New Technologies; IEEE Control System Soc; Lab-STA; ATTNA; LAGIS; CReSTIC; UNIV CATHOLIQUE DE LILLE; LIAS; MIS; ENIS; Univ Poitiers, 2015. ISBN 978-1-4799-8318-6.

[80] Y. Cai, X. Guo, Z. Zhong, and W. Mao. Dynamic meshing for deformable image registration. *Computer-Aided Design*, 58(SI):141–150, JAN 2015. ISSN 0010-4485. doi: 10.1016/j.cad.2014.08.009.

[81] E. Smistad, M. Bozorgi, and F. Lindseth. Fast: framework for heterogeneous medical image computing and visualization. *International Journal of Computer Assisted Radiology and Surgery*, 10(11):1811–1822, NOV 2015. ISSN 1861-6410. doi: 10.1007/s11548-015-1158-5.

[82] C. A. S. J. Gulo, H. F. de Arruda, A. F. de Araujo, A. C. Sementille, and J. M. R. S. Tavares. Efficient parallelization on GPU of an image smoothing method based on a variational model. *Journal of Real-Time Image Processing*, Jul 2016. ISSN 1861-8219. doi: 10.1007/s11554-016-0623-x.

[83] T.-A. Nguyena, A. Nakib, and H.-N. Nguyen. Medical image denoising via optimal implementation of non-local means on hybrid parallel architecture. *Computer Methods and Programs in Biomedicine*, 129:29 – 39, 2016. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2016.02.002.

[84] H. Koestler, M. Stuermer, and T. Pohl. Performance engineering to achieve real-time high dynamic range imaging. *Journal of Real-Time Image Processing*, 11(1):127–139, JAN 2016. ISSN 1861-8200. doi: 10.1007/s11554-012-0312-3.

[85] J. Hu, X. Zhao, and H. Zhang. A GPU-based multi-resolution approach to iterative reconstruction algorithms in X-ray 3D dual spectral computed tomography. *Neurocomputing*, 215(SI):71–81, NOV 26 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2016.01.115.

[86] X. Du, J. Dang, Y. Wang, S. Wang, and T. Lei. A parallel nonrigid registration algorithm based on b-spline for medical images. *Computational and Mathematical Methods in Medicine*, 2016, 2016. doi: 10.1155/2016/7419307.

[87] J. L.-R. D. B. Heras, F. Arguello, D. Kainmueller, S. Zachow, and M. Boo. GPU-accelerated level-set segmentation. *Journal of Real-Time Image Processing*, 12(1):15–29, 2016. ISSN 1861-8200. doi: 10.1007/s11554-013-0378-6.

[88] J. Chen, S. Zhou, and H. Min. Parallel delay-and-sum algorithm implemented on supervessel cloud with high-performance fpga. pages 1015–1020. Institute of Electrical and Electronics Engineers Inc., 2016. doi: 10.1109/HPCC-SmartCity-DSS.2016.0144.

[89] M. Riegler, M. Lux, C. Griwodz, C. Spampinato, T. De Lange, S. Eskeland, K. Pogorelov, W. Tavanapong, P. Schmidt, C. Gurrin, D. Johansen, H. Johansen, and P. Halvorsen. Multimedia and medicine: Teammates for better disease

detection and survival. pages 968–977. Association for Computing Machinery, Inc, 2016. doi: 10.1145/2964284.2976760.

[90] W.-M. Pang, K.-S. Choi, and J. Qin. Fast gabor texture feature extraction with separable filters using GPU. *Journal of Real-Time Image Processing*, 12(1):5–13, JUN 2016. ISSN 1861-8200. doi: 10.1007/s11554-013-0373-y.

[91] T. Wang, K. Celik, and A. K. Somani. Characterization of mountain drainage patterns for gps-denied uas navigation augmentation. *MACHINE VISION AND APPLICATIONS*, 27(1):87–101, JAN 2016. ISSN 0932-8092. doi: 10.1007/s00138-015-0723-9.

[92] A. Sabne, X. Wang, S. Kisner, C. Bouman, A. Raghunathan, and S. Midkiff. Model-based iterative CT image reconstruction on GPUs. pages 207–220. Association for Computing Machinery, 2017. doi: 10.1145/3018743.3018765.

[93] M. Jaros, P. Strakos, T. Karasek, L. Riha, A. Vasatova, M. Jarogova, and T. Kozubek. Implementation of K-means segmentation algorithm on Intel Xeon Phi and GPU: Application in medical imaging. *Advances in Engineering Software*, 103:21–28, JAN 2017. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2016.05.008.

[94] K. D. Toennies. *Digital Image Acquisition*, pages 21–82. Springer London, London, 2012. ISBN 978-1-4471-2751-2. doi: 10.1007/978-1-4471-2751-2_2.

[95] M. Birk, M. Zapf, M. Balzer, N. Ruiter, and J. Becker. A comprehensive comparison of GPU- and FPGA-based acceleration of reflection image reconstruction for 3D ultrasound computer tomography. *Journal of Real-Time Image Processing*, 9(1, SI):159–170, MAR 2014. ISSN 1861-8200. doi: 10.1007/s11554-012-0267-4.

[96] R. Sharma and A. Sharma. Segmentation methods in atherosclerosis vascular imaging. *Journal Informatica Medica Slovenica*, 11:52–69, 2006. ISSN 1318-2129.

[97] Y. Zhuge, K. C. Ciesielski, J. K. Udupa, and R. W. Miller. GPU-based relative fuzzy connectedness image segmentation. *Medical Physics*, 40(1), JAN 2013. ISSN 0094-2405. doi: 10.1118/1.4769418.

[98] J. Mertes, N. Marranghello, and A. Pereira. Real-time module for digital image processing developed on a FPGA. volume 12, pages 405–410, 2013. doi: 10.3182/20130925-3-CZ-3023.00072.

[99] T. Ustun, N. Iftimia, R. Ferguson, and D. Hammer. Real-time processing for fourier domain optical coherence tomography using a field programmable gate array. *Review of Scientific Instruments*, 79(11), 2008. doi: 10.1063/1.3005996.

[100] A. Nieto, V. Brea, D. L. Vilarino, and R. R. Osorio. Performance analysis of massively parallel embedded hardware architectures for retinal image processing. *EURASIP Journal on Image and Video Processing*, 2011. ISSN 1687-5281. doi: 10.1186/1687-5281-2011-10.

[101] P. B. Nguyen, J.-O. Park, S. Park, and S. Y. Ko. Medical micro-robot navigation using image processing - blood vessel extraction and x-ray calibration. In *2016 6TH IEEE INTERNATIONAL CONFERENCE ON BIOMEDICAL ROBOTICS AND BIOMECHATRONICS (BIOROB)*, Proceedings of the IEEE RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, pages 365–370. IEEE, 2016. ISBN 978-1-5090-3287-7.

[102] F. Gebali. *Algorithms and Parallel Computing*. John Wiley & Sons, 2011. ISBN 978-0-470-90210-3.

[103] D. Page. *A Practical Introduction to Computer Architecture*. Springer, 2009. ISBN 978-1-84882-255-9. doi: 10.1007/978-1-84882-256-6.

Part B - Article 2

Efficient parallelization on GPGPU of an image smoothing method based on a variational model

Abstract

Medical imaging is fundamental for improvements in diagnostic accuracy. However, noise frequently corrupts the images acquired, and this can lead to erroneous diagnoses. Fortunately, image preprocessing algorithms can enhance corrupted images, particularly in noise smoothing and removal. In the medical field, time is always a very critical factor, and so there is a need for implementations which are fast and, if possible, in real time. This study presents and discusses an implementation of a highly efficient algorithm for image noise smoothing based on general purpose computing on graphics processing units techniques. The use of these techniques facilitates the quick and efficient smoothing of images corrupted by noise, even when performed on large-dimensional data sets. This is particularly relevant since GPU cards are becoming more affordable, powerful and common in medical environments.

Keywords: GPGPU, CUDA, Image processing, Multiplicative Noise

1 Introduction

Computational image processing is a field that has seen tremendous advances in recent years. These advances are the result of huge demands coming from areas such as medicine [1], agriculture [2], security [3], traffic and satellite data analysis [4] and industry [5]. These fields require image processing tasks such as noise and artifact removal and smoothing [6], geometrical correction [7], contrast enhancement [8], image restoration [9], and illumination correction [10]. Briefly, the use of image processing techniques, particularly of image preprocessing, is mainly intended to enhance the data presented in the original images so that the processed data can be analyzed more easily using higher-level techniques of computational image analysis, such as image segmentation [11] or image registration [12]. However, many of the original images that need to be enhanced have large dimensions [13] and need to be processed in real time or near real time [14]. This is the case, for example, in the fields of robotic navigation or assisted surgery, or even when the input data are long sequences of 2D or 3D images, such as in ultrasound imaging [15]. Additionally, to obtain more robust and efficient results, the computational complexity of the more recent methods has considerably increased, leading to slower runtimes. Therefore, the use of parallel computing strategies has attracted attention, and this has led to higher speeds, particularly in time-constrained applications for medical diagnosis [16].

Frequently, noise corrupts images, and this may be due to the image acquisition procedure involved or to artifacts generated by data transmission or other processes [17]. The image smoothing method proposed by Jin and Yang [18] has obtained very promising results, particularly when applied to medical images. However, a long computational time when performing several iterations on the input image is required by this method, especially when applied to large-scale images; as a result, its use has become less attractive for some potential applications. Additionally, there is a frequent and increasing demand for fast responses from computational methods in high-resolution image processing, and real time is preferable due to the severe time constraints that characterize medical imaging.

Therefore, we have developed a parallel implementation of the smoothing method proposed by Jin and Yang [18] using general purpose computing on graphics processing units (GPGPU) [19] and compute unified device architecture (CUDA) [20] in order to speed up its runtime. We have assessed the performance of this strategy by comparing the runtime of parallel implementation (GPU) against that of sequential implementation (CPU).

The method adopted for image smoothing selects each pixel from the input image and thus requires a large number of calculations; this leads to the long runtime mentioned. Briefly, the method involves the use of an $(m \times n)$ matrix, which is processed for T iterations. Thus, the computational complexity of the processing of the input image is equal to $O(m \times n \times T)$, where m and n are the number of rows and columns of the input image, respectively.

In our parallel implementation, the input image data are stored in the GPU's memory, where the highest number of accesses occurs, in order to eliminate as many data accesses as possible within the main memory system [19, 21]. Hence, input image processing is executed in parallel in the GPU. The experimental findings confirmed that the combination of the CUDA architecture and GPGPU techniques was very promising in terms of speeding up the runtime of image processing and computational analyses. These approaches led to high processing performance at a low cost, mainly when compared with parallel implementations in multicomputers.

As far as the authors known, this was the first time that the smoothing method adopted was parallelized using CUDA architecture and GPGPU techniques. The findings are of great interest for image processing and analysis, mainly within the medical community. In this case, medical images of ever higher resolution need to be smoothed as fast as possible in real clinical scenarios. Nowadays, computers with GPUs are commonly available in medical environments and, although these computers are not always the most up-to-date models, their computational power is still sufficient to achieve efficient fast results.

This paper is organized as follows: Sect. 2 introduces the image smoothing method proposed by Jin and Yang [18]; Sect. 3 describes the metrics of structural similarity (SSIM), peak signal-to-noise ratio (PSNR) and normalized cross-correlation (NCC), all of which are used to assess the quality of the smoothing results. Sect. 4.3 presents the parallel implementation of the image smoothing method; the computational runtimes demanded by the CPU- and GPU-based implementations are discussed in Sect. 5; and finally, Sect. 6, presents the concluding remarks.

2 Image smoothing method

Images frequently have multiplicative noise, which comes from multiplying an original image I by a noisy image I_n [22]. This type of noise is present, for example, in microscopy, ultrasound and infrared imaging [23]. Multiplicative noise is usually more difficult to remove than additive noise [24]. Therefore, to overcome this problem, variational models for multiplicative noise removal have been integrated into smoothing methods specially developed for such images [24, 25]. In 2011, Jin and Yang [18] proposed a very promising method for removing and smoothing multiplicative noise from corrupted images using the variational model for additive noise removal proposed by Rudin et al. [26], as shown here:

$$min_u \left\{ J(u) + \lambda \int_\Omega (f-u)^2 \right\}, \tag{1}$$

where Ω is a closed area belonging to R^2, f is the image corrupted by additive noise, u is the image in the current smoothing iteration, $J(u)$ is a regulator term and λ is a weight parameter. Jin and Yang designed the method specifically to remove multiplicative noise from ultrasound images, and they concluded that the function proposed by Krissian et al. [27] could be adopted to solve the variational model of Eq. 1, using:

$$E(u) = \int_\Omega \frac{(f-u)^2}{u}, \tag{2}$$

where u is the original image, $f = u + \sqrt{ug}$ is now the input image corrupted by multiplicative noise and g is a Gaussian variable with a nonzero mean. Thus, the variational model adopted by Jin and Yang [18] is:

$$min_u \left\{ J(u) + \lambda \int_\Omega \left[\frac{(f-u)^2}{u} \right] \right\}, \tag{3}$$

where $\lambda > 0$ is a weight parameter. As such, the model given by Eq. (3) deals with the problem of multiplicative noise removal by adopting:

$$\partial_t u = div\left(\frac{\nabla u}{|\nabla u|}\right) + \lambda\left(\frac{f^2}{u^2} - 1\right), \tag{4}$$

where ∇ and div are the gradient and divergent operators, respectively. In order to discretize the continuous part of Eq. (4), Rudin et al. [26] used the finite difference scheme, adopting $h = 1$ for the step size and Δt for the time interval, which leads to:

$$\begin{aligned}
A &= D_{\underset{x}{+}}(u_{i,j}) = u_{i+1,j} - u_{i,j},\\
B &= D_{\underset{x}{-}}(u_{i,j}) = u_{i,j} - u_{i-1,j},\\
C &= D_{\underset{y}{+}}(u_{i,j}) = u_{i,j+1} - u_{i,j},\\
D &= D_{\underset{y}{-}}(u_{i,j}) = u_{i,j} - u_{i,j-1},\\
|D_x(u_{i,j})| &= \sqrt{A^2 + \left(m\left[C,D\right]\right)^2 + \delta},\\
|D_y(u_{i,j})| &= \sqrt{C^2 + \left(m\left[A,B\right]\right)^2 + \delta},
\end{aligned} \tag{5}$$

where the parameter $\delta > 0$ is a constant defined close to zero, and term m is defined as:

$$m[a,b] = \frac{sign(a) + sign(b)}{2} min(|a|,|b|), \tag{6}$$

where $min(|a|,|b|)$ is a function that returns the smallest absolute value between a and b and $sign(a)$ is a function that determines the sign of a, returning 1 if a is positive, -1 if it is negative and 0 if a is equal to 0. Assuming the iterations of the model $k = 1,2,...$, Eq. (4) can be rewritten as:

$$\begin{aligned}
u_{i,j}^{n+1} =& \delta t\left[\frac{-D_{\underset{x}{+}}(u_{i-1,j}^n) + D_{\underset{x}{+}}(u_{i,j}^n)}{-|D_x(u_{i-1,j}^n)| + |D_x(u_{i,j}^n)|} + \right.\\
&\left.\frac{-D_{\underset{y}{+}}(u_{i,j-1}^n) + D_{\underset{y}{+}}(u_{i,j}^n)}{-|D_y(u_{i,j-1}^n)| - |D_y(u_{i,j-1}^n)|}\right]\\
&+ \lambda^n\left(\frac{f^2}{(u_{i,j}^n)^2} - 1\right) + u_{i,j}^n,
\end{aligned} \tag{7}$$

Figure 1: Original, noisy and smoothed (128x128 pixels) images, respectively

where f is the input image affected by multiplicative noise. In this method, the λ parameter automatically calculated for each new iteration as:

$$\lambda^n = \frac{1}{\sigma^2}\left(\Sigma_{i,j}\left[\left(D_{\bar{x}}\left(\frac{D_{\overset{+}{x}}(u^n_{i,j})}{|D_x(u^n_{i,j})|}\right) + D_{\bar{y}}\left(\frac{D_{\overset{+}{y}}(u^n_{i,j})}{|D_y(u^n_{i,j})|}\right)\right) \frac{(u^n_{i,j} - f)u^n_{i,j}}{u^n_{i,j} + f}\right]\right), \tag{8}$$

where σ^2 is the variance of the image at iteration k. As an example, Fig. 1 shows the result of the smoothing method when applied to ultrasound images.

3 Assessment metrics

The comparison between two images is a natural task for the human visual system, but it is not so natural for computer systems. Therefore, various authors have proposed different solutions which assess the similarities between two images and, in particular, evaluate the performance of image preprocessing methods [28–32]. Basically, there are two classes of solutions: One is based on intensity error and the other on structural information.

3.1 Based on intensity error

For image smoothing, the comparative solutions or similarity indices use intensity error in order to estimate the error between the enhanced image, i.e., the smoothed image, and the original image before noise corruption. The main disadvantage of these similarity indices is the possibility of failure where there are displacements between the images under comparison. Moreover, these indices compare the intensity variation of each pixel of the input images, which can lead to similar results for images with different types of geometrical distortions [29]. Nevertheless, indices based on intensity error are frequently

used to compare the performance of image enhancement [33–35] and smoothing [17, 36] methods, due to their simplicity.

In particular, the peak signal-to-noise ratio (PSNR) index has been widely used to assess the performance of image restoration and smoothing methods. This index determines the ratio between the highest possible strength of a signal, which in the case of images is the highest intensity value, and its strength as affected by noise [15, 17]. For simplicity, the PSNR is represented according to a logarithmic scale (base 10), since some signals can have very high values.

The PSNR can be calculated from the mean squared error (MSE), which is computed as:

$$MSE = \frac{1}{m \quad n} \sum_{i=0}^{m-1} \sum_{j=1}^{n-1} [I(i,j) - I_r(i,j)]^2 , \tag{9}$$

where m and n are the dimensions of the images I and I_r to be compared, as follows:

$$PSNR = 10log_{10}\left(\frac{MAX_I^2}{MSE}\right), \tag{10}$$

where MAX_I is the maximum intensity value that a pixel can assume, which is equal to 255 for 8-bit grayscale images. Thus, the higher the PSNR value is, the more efficient the performance of the preprocessing algorithm is. The two images are considered identical, when the MSE value is 0 (zero), and the PSNR value is undefined.

Normalized cross-correlation (NCC) is another metric based on pixel intensity. It is widely used in image registration [37] to compare the degree of similarity between two input images. NCC is as follows

$$NCC = \frac{\sum_{i=1}^{m \times n} x_i y_i}{\sqrt{\sum_{i=1}^{m \times n} x_i^2 \sum_{i=1}^{m \times n} y_i^2}}, \tag{11}$$

where x_i and y_i denote the intensity values of each pixel of the $(m \times n)$ images under comparison, leading to values in the interval $[0,1]$, where 1 (one) indicates a best match [38].

3.2 Based on structural information

In this class of quality metrics, the goal is to find changes in the structural information of the images under comparison. The analysis of the structural information represented in the input images assumes that the human vision system is adapted to extract, i.e., segment, structural information from what is seen, and to search for changes in the structures

detected. In other words, any possible differences, such as those due to artifacts generated by noise processes [39], are quantified.

The structural similarity index (SSIM) is the main index in this category which analyzes the performance of computational image processing methods [36, 40]. Wang et al. [29] proposed this similarity index in an attempt to prevent images with very different visual qualities ending up with high similarity values, as can happen when the similarity indices are based on intensity error. The index measures the change in three components of each image under comparison: luminance, contrast and image structure. The former is defined as average pixel intensity. The contrast component is modeled using the standard deviation of the intensity, while image structure comes from the normalized image using the standard deviation of images under comparison. The SSIM is as follows:

$$SSIM(x,y) = l(x,y)^{\alpha} \cdot c(x,y)^{\beta} \cdot s(x,y)^{\gamma}, \tag{12}$$

where l refers to luminance, c to contrast and s to structure, and $\alpha > 0$, $\beta > 0$ and $\gamma > 0$ are weights. These three components are relatively independent, and therefore, modifying one of them does not affect the others. More details of the calculation of these components, as well as a detailed analysis of them, are presented in [29].

The SSIM is an index which applies to each pixel of the input image, and for convenience, the mean SSIM is usually adopted. The mean structure similarity index (MSSIM) is the average of all the SSIM values obtained. For identical images, this index is equal to 1 (one). As the images become different, the index becomes lower until it is equal to -1 when the images are exactly opposite, i.e., one is the negative of the other.

4 Parallelization of the smoothing method

Studies have shown that GPU-based parallel methods have focused on massively parallel programming [41], and most common image processing methods can operate with parallelization strategies based on the data decomposition technique. This section describes the steps involved in the GPU-based parallel implementation, which was developed in order to optimize the runtime performance of the adopted smoothing method.

The smoothing method adopted in this study, as described in Sect. 2, made use of four fundamental equations to find a solution for the multiplicative noise smoothing process give by Eq. (4). The method starts by solving the finite difference scheme adopted in Eq. (5). Then, Eq. (7) obtains the final value for each pixel according to the ongoing iteration, and Eq. (8) finds the associated weight parameter. Thus, Eqs. (4), (5), (7) and (8) define

a sequence of steps for the parallelization of the smoothing method. The implementation procedure was based on the NVIDIA programming best practices guide [19].

The CUDA architecture was developed with the objective of using data parallelism, by establishing a new model named single instruction multiple thread (SIMT). In this model, data are represented as a stream, which is structured as an array, and when running one or more instructions using this array, the instructions are defined as a kernel [16, 19]. A kernel performs operations in parallel along the entire stream, using it as both input and output [19, 42].

In the SIMT model, the calling of multiple kernels follows a hierarchy of thread groups. This feature divides each kernel into independent blocks, and as a result, the efficient threading support in the GPUs ensures transparency, portability and scalability, besides allowing a CUDA program to be executed in any number of processor cores. Threads are used for fine-grained parallelism; groups of threads, defined as "blocks", are

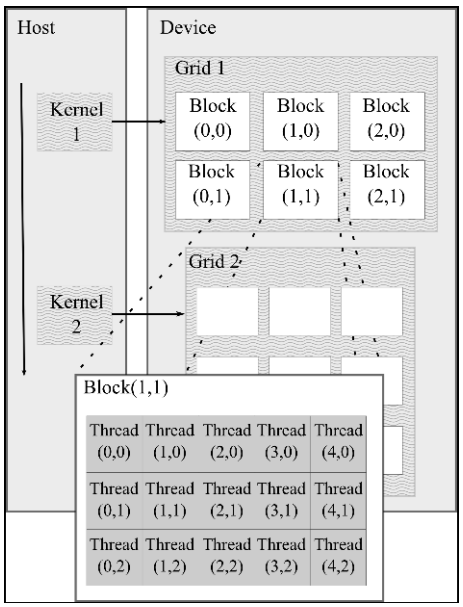


Figure 2: Representation of the single-instruction multiple-thread model (adapted from [19])

used for coarse-grained parallelism; groups of blocks are placed in a grid which represents a kernel call. As illustrated in Fig. 2, this hierarchy allows each thread within a block and each block in a grid to have a unique identifying index [42].

4.1 Setting the occupancy level

The setting of each kernel must be adjusted to use the correct number of blocks and threads in order to optimize the occupancy of the CUDA cores (code lines in Fig. 3);

i.e., if the number of blocks and threads is not sufficient, some cores will not be able to execute the code, wasting some of the processing power. In our implementation, we used 256 threads per block in all the kernels. The number of blocks B is given by:

$$B = \frac{T_{px} + T_{Tb} - 1}{T_{Tb}}, \tag{13}$$

where T_{px} is the total number of pixels of the input image and T_{Tb} is the number of threads per block. In this case, we defined one two-dimensional variable (*numBlocks*), which has the image height (*HImage*) as the first dimension (T_{px}) and the image width as the second dimension (T_{px}). These calculations determine the settings used to perform one thread per pixel. When there are excess threads, a stopping criterion discards them.

```
1 dim3 threadsPerBlock(16, 16)
2 dim3 numBlocks((HImage + 15) / threadsPerBlock.y,
3 (WImage + 15) / threadsPerBlock.x)
```

Figure 3: Definitions for the settings of each kernel used in the experiments

Applications developed for massively parallel architectures achieve greater performance when the graphics card resources are used efficiently. The occupancy level of the GPU measures the proportion of active processors in the graphics card during a kernel execution. This calculation takes into account the following specification query attributes acquired from the CUDA device: the maximum number of threads per block, the number of blocks per multiprocessor, the number of registers per multiprocessor and the shared memory per multiprocessor. Increasing the number of concurrent threads is a good strategy for the purpose of making full use of the GPU, and the limit of threads is defined by the architecture. However, a high level of GPU occupancy does not guarantee an additional performance gain [19] because there is a problem of memory latency, and a high level of occupancy may reduce the overall performance [43].

4.2 Optimizing the memory hierarchy in CUDA

As shown in Fig. 4, each multiprocessor can use four types of memory: a set of registers for each stream multiprocessor (SM), a shared memory between the SMs, a constant cache shared between the SMs, and a texture cache which optimizes the bandwidth of the texture memory. Registers have the largest bandwidth, and like other kinds of memory, threads can access them; threads can also access data in different memory spaces. Each SM used in the experiments has 256 kB worth of memory registers [19].

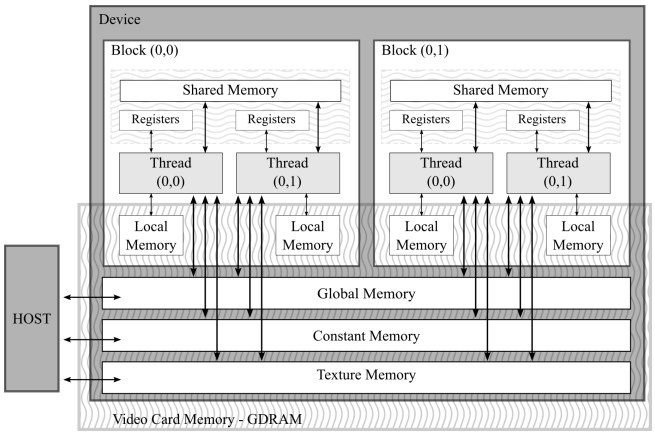


Figure 4: Memory spaces accessed by each thread (adapted from [41])

In the case of shared memory, the bandwidth is similar to the registers, and threads can cooperate to load and compute data shared by them. Each memory module has a set of 32-bit registers, which makes the threads access consecutive positions of a data vector more efficiently. A module can receive multiple requests for the same data, but this creates conflicts. However, automatic serialization satisfies all memory access requests. As this serialization can reduce bandwidth performance, a broadcast device is set up to prevent the reading of all the threads at the same memory address [19]. On the other hand, all threads can access the GPU global memory (GDRAM) simultaneously.

However, there are some restrictions which improve the bandwidth. Global memory has the lowest bandwidth but has the largest storage capacity. In order to obtain the maximum possible speedup, a group of threads are used which has consecutive indices and is bundled into a unit named a warp. Thus, a single SM can run multiple warps simultaneously. The size of a warp depends on the GPU specification [19, 42].

All threads have read-only access to the GPU memory cache, which has 48 kB for each SM; moreover, the threads of a half-warp can read only one memory address. Only instructions from the GPU can write into this kind of memory, and these processes persist throughout the execution of multiple kernel calls [19].

All threads can also access the texture memory, which is only read by kernels. This kind of memory uses a separate cache with a capacity of 32 kB per SM and provides high-performance accesses when all threads perform operations on memory addresses close to them [43]. The types of access of on-chip memory for Compute 3.5 and later devices are indicated in Table 1.

4.3 Implementation of kernels in CUDA C

Tasks of computational image processing and analysis usually involve a large amount of data processing. Thus, the first strategy is to allocate the required space in the GDRAM and then copy the input image as a data matrix from the host memory (RAM) to the device's memory (GDRAM); this process allows data to be managed directly in the GPU. Accesses to the coalesced memory are performed in contiguous segments; half-warps access the segments simultaneously. Such accesses are known as coalesced memory accesses, and they enable parallel operations, thereby reducing the number of memory transactions [19, 42]. The data are then loaded into contiguous segments, and this allows a thread block to process an input image more efficiently; moreover, both the global memory and the texture memory are used.

Table 1: Types of memory access in CUDA [44]

Memory	Location	Access	Cached	Scope
Register	INT	r/w	No	One thread
Local	INT	r/w	Yes	One thread
Shared	INT	r/w	N/A	All threads in a block
Global	EXT	r/w	Yes	All threads + host
Constant	EXT	r	Yes	All threads + host
Texture	EXT	r/w	Yes	All threads + host

r Reading access, *w* writing access, *INT* internal memory space location, *EXT* external memory space location

Equations (5), (7) and (8) were implemented in the kernels called *kDiFinitas, kVariancia* and *kFinal*, respectively. The threads from the *kDiFinitas* kernel perform the computations in Eq. (5) in each image pixel independently. This kernel has several threads, each of which represents a matrix index and processes a specific image pixel. Thus, to manage access to a set of image pixels in the "for-loops", each pixel has an access condition.

First of all, in the *kDiFinitas* kernel, each pixel from the input image is associated with a thread, and then the thread blocks are stored in the texture memory. After running the *kDiFinitas* kernel, the *kVariancia* kernel performs the parallelized computation of the λ parameter according to Eq. (7). In the parallel implementation, an auxiliary vector stores the values of the operations involved in each iteration, i.e., each thread calculates the resultant value of each iteration. Figure 5 presents the pseudocode of the developed algorithm.

The *kFinal* kernel computes the weight parameter, used previously in the *kVariancia* kernel, and then applies it to each image pixel, giving access to the texture cache and

```
1  Input: Noisy image
2  /* Host program executed on CPU */
3  Allocate CPU and GPU memory
4  Store image to CPU memory
5  Copy image from CPU memory to GPU memory
6  Set the number of threads per blocks
7  /* kDiFinitas: Kernel program executed on each thread block */
8  Parallel each image pixel
9  Compute the finite difference using Eq.(5)
10 /* kVariancia: Kernel program executed on each thread block */
11 Parallel each image pixel
12 Compute weight parameter by Eq.(8)
13 Call kFinal kernel
14 /* kFinal: Kernel program executed on each thread block */
15 Parallel each image pixel
16 Create a new vector with zeros to store the smoothed image
17 Compute the new pixel values for each new iteration using Eq.(7)
18 Copy image from GPU memory to CPU memory
19 Output: Smoothed image
```

Figure 5: Pseudocode of the developed parallel implementation

coalesced access to the global memory. Each thread attributes the resulting value to the corresponding memory, providing the data needed to calculate the final sum of each image pixel. Finally, the *SomaElem* kernel assists with the calculation of the vector values. The vector is divided into two equal parts, their values are summed, and each thread sums two values and keeps them in the lowest available vector position. This procedure continues until only one vector position remains for the storage of the resulting sum. In the case of a vector with an odd number of elements, an extra element with a zero value is used. This procedure uses the partitioning strategy of the global memory to optimize the bandwidth of the active warps during memory access; the warps are organized into partitions. This is the slowest kernel used, and this is because the memory blocks become less contiguous while the elements are processed. Figure 6 illustrates the implemented parallelization technique.

An image corrupted by multiplicative noise is used as input (step 1), and after running the kernels described previously in steps 4-8, the result will be a new noise-smoothed image. Steps 4, 6, and 7 perform the reading of data in the texture memory. On the other hand, the results of each step of memory writing go into the global memory, where the output images are stored.

Equations (5), (7) and (8) were implemented as a nested “for-loop”. A CPU-based implementation was also developed as a comparison with the GPU-based implementation. The main memory system was accessed contiguously for all of these loops in order to optimize execution, and the GDRAM was accessed contiguously as well, creating a fair comparison [19] between the implementations.

5 Experiments and discussion

This section describes the infrastructure used to perform the experiments and also discusses the results.

5.1 Test Infrastructure

The used test infrastructure includes a desktop computer equipped with an Intel(R) Core(TM) i7-4790 3.60 GHz processor, 16 GB of RAM (DDR3-1600 MHz), Linux Ubuntu 14.04 operating system, CUDA [9] nvcc release 7.5 compiler driver and GNU gcc/g++ compiler version 4.8.4. Additionally, there was a GPU NVIDIA Tesla K20c, with 2496 CUDA cores and 5 GB of GDRAM.

5.2 Results and Discussion

In this section, we present results of experiments aimed at evaluating the performance of the method adopted. The runtime performance of GPU-based implementation is the focus of this study; however, the PSNR, SSIM, and NCC metrics were used to confirm the smoothing method's accuracy. In the tests, 15, 25 and 50 smoothing iterations were adopted.

Table 2: Comparison between the computational time (in milliseconds) required by the CPU- and GPU-based implementations to smooth the test static images with 50 iterations

Images	Tesla	CPU
128×128	30.41 ± 0.81	14.08 ± 0.10
256×256	36.72 ± 0.20	56.37 ± 0.26
512×512	59.04 ± 0.96	225.90 ± 1.17
1024×1024	133.38 ± 1.86	944.99 ± 18.34
2048×2048	423.94 ± 1.23	3761.56 ± 32.18
4096×4096	1617.16 ± 7.09	$15,180.35 \pm 26.22$

We used a set of six images with different resolutions (128×128, 256×256, 512×512, 1024×1024, 2048×2048 and 4096×4096 pixels), built synthetically with an image editor software and then corrupted with synthetic multiplicative noise of a variance equal to 0.3. There were 100 iterations for each test, and the mean and the standard deviation values of the time spent smoothing each input image were calculated. The total time spent (Table 2) was computed from the moment the data were loaded into the main memory

[9] CUDA compiler and development suite are available to download through the NVIDIA Web site `https://developer.nvidia.com/cuda-downloads`.

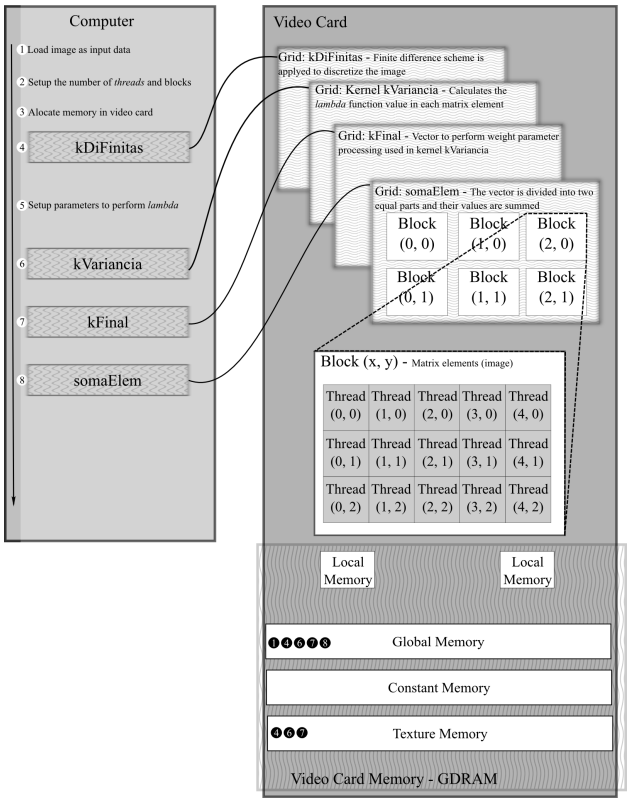


Figure 6: Parallel CUDA-based implementation of the adopted image smoothing method

system until the end of the smoothing process when the resultant image was produced. The function *cudaThreadSynchronize* was performed after each kernel call, forcing the CPU to wait for the complete kernel execution, and the *sdkResetTimer*, *sdkStartTimer* and *sdkStopTimer* timing functions were used to obtain the kernel execution time. The execution times of each kernel were added together to obtain the total execution time. Table 2 shows that the execution times of the CPU-based implementation were longer than those of the GPU-based implementation except in the case of the smallest test image (128×128 pixels). This distinct behavior occurred because the speedup achieved with the data processed in the CUDA cores did not justify the computational effort involved in transferring a small amount of data to the GPU memory or the latency times necessary for the initialization of the GPU. For larger images, the speedup of the GPU was around 10, but less for smaller ones. Moreover, the GPU-based implementation achieved noise smoothing in real time for all tested images. The parallel implementation had transparent and portable scalability in GPUs based on CUDA architecture; besides, the performance scales increased exponentially, as shown in Fig 7. Furthermore, when we considered images with dimensions greater than 256×256 pixels, a speedup of the GPU-based implementation was evident; for example, it was about 10.65 times faster for images

with 4096×4096 pixels.

As an illustrative example, Fig. 8 shows the results of the CPU- and GPU-based implementations applied to the test images. Figure 8 shows, from left to right, the image affected by the multiplicative noise and the images smoothed by the CPU- and GPU-based implementations.

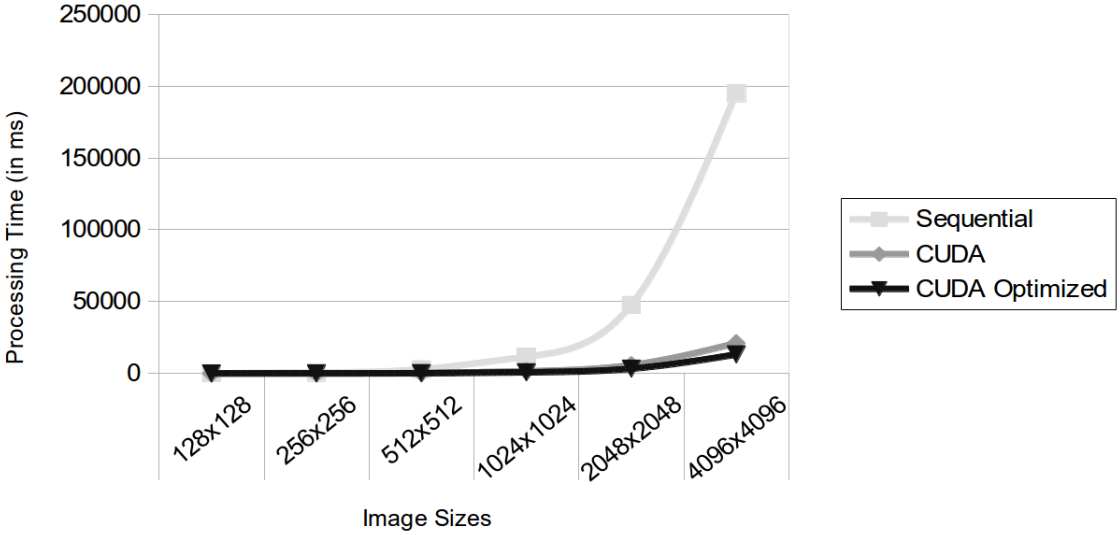


Figure 7: Processing time of the proposed GPU-based implementation, which scales up exponentially

Table 3: NCC and SSIM values computed for the static test images using 15, 25 and 50 iterations

Images	NCC			SSIM		
	15	25	50	15	25	50
128×128	0.99999	0.99996	0.99962	0.99992	0.99869	0.98652
256×256	0.99973	0.99884	0.99752	0.99894	0.99612	0.98383
512×512	0.99667	0.99679	0.99554	0.99959	0.99932	0.99715
1024×1024	0.99664	0.99696	0.99587	0.99967	0.99967	0.99914
2048×2048	0.99819	0.99827	0.99798	0.99996	0.99995	0.99988
4096×4096	0.99889	0.99894	0.99864	0.99999	0.99999	0.99997

The values listed in Table 3 were computed using NCC and SSIM metrics in order to confirm that the structural information resulting from the noise images corresponded to smoothed images, since all values were close to 1 (one). As for the smoothing method's accuracy and time performance, the optimal number of iterations for better image preservation seems to be 15.

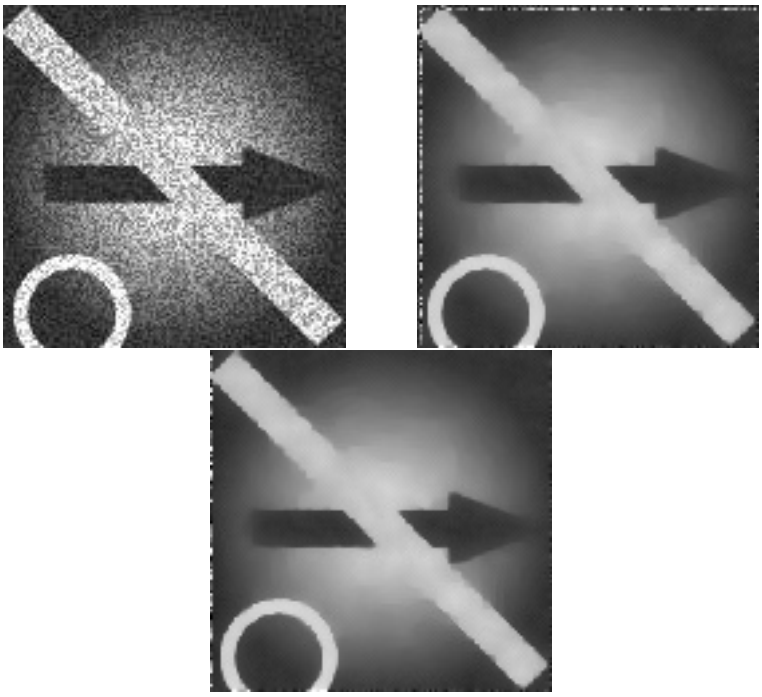


Figure 8: Original noisy test image with 4096×4096 pixels and the smoothed images obtained by the CPU- and GPU-based smoothing implementations, respectively

The PSNR values were also computed for each static test image before and after being smoothed by the CPU- and GPU-based implementations (Table 4). The values demonstrated the efficiency of the smoothing method and confirmed that the two implementations smoothed the images using the method adopted.

We also tested three synthetic videos with 240 frames and different resolutions (128×128, 256×256, 512×512) and one real ultrasound video with 255 frames of 320×240 pixels. The smoothing method was applied only once for each video frame.

The average runtime for the real ultrasound video was 5.92 s for the CPU-based implementation and 2.87 s for the parallel implementation in CUDA. Thus, the processing time of the parallel implementation was about 2.06 times faster when processing the entire ultrasound video. Figure 9 shows an example of the smoothing of a video frame selected randomly from the tested video.

Table 5 indicates the frame rates of the CPU- and the GPU-based implementations when smoothing the four test videos. In this table, the values in bold can be considered in line with real-time processing (>20 frames per second) and therefore acceptable for routine medical image processing [44, 45]. As given in Table 5, the experiments using the parallel GPU-based implementation revealed an even higher reduction in the runtime of the smoothing method in relation to the CPU-based implementation, confirming initial

Table 4: PSNR values computed for the static test images before (noisy) and after being smoothed by the CPU- and GPU-based implementations using 15, 25 and 50 iterations

Images	PSNR						
	Noisy	GPU smoothed			CPU smoothed		
		15	25	50	15	25	50
128×128	+16.71226	+26.45354	+28.01845	+27.08323	+26.45101	+28.01758	+27.08876
256×256	+13.63118	+21.45840	+21.15450	+20.48712	+21.46481	+21.13883	+20.51067
512×512	+10.71005	+11.70306	+11.84286	+12.18230	+11.70372	+11.84272	+12.18366
1024×1024	+11.04700	+14.64996	+14.63267	+14.81482	+14.64584	+14.63146	+14.80796
2048×2048	+10.39326	+13.64743	+13.73511	+14.05504	+13.64635	+13.73472	+14.05501
4096×4096	+9.92586	+13.15495	+13.30716	+0.99864	+0.99999	+13.30648	+13.67202

expectations.

Table 5: Frames per second (FPS) rate obtained in the CPU- and GPU-based implementations with the smoothing method applied with 15, 25 and 50 iterations

Video Resolution	Total of FPS rate obtained					
	GPU			CPU		
	15	25	50	15	25	50
128x128	**116.60**	**69.91**	**34.12**	**249.79**	**152.93**	**76.78**
256x256	**94.23**	**57.16**	**28.84**	**52.73**	**33.10**	16.39
320x240	**88.61**	**54.00**	**27.93**	**48.16**	**32.20**	16.81
512x512	**57.79**	**37.10**	19.59	13.75	8.44	4.21

The CUDA architecture as a computational infrastructure for image preprocessing has revealed to be a viable, capable and alternative option to deliver high-performance processing in many applications; moreover, it can even provide real-time processing at an affordable **cost**. Here, the performance gain of the parallel GPU-based implementation confirms the high processing capacity available in the CUDA architecture, with all videos used in the experiments processed in real time. Today, the available resource in these graphics cards have increased the performance gain more efficiently, taking into consideration the number of cores and GDRAM memory as well as the SIMT parallel model associated with memory optimization techniques.

Therefore, the benefit of using GPU-based implementations can be totally justified since the reduction in the runtime can minimize or even eliminate the time restrictions; such restrictions are common in many applications (such as in the medical field) that use image processing and analysis methods, requiring fast or real-time results for image-based [4, 46]. However, optimal implementation requires maximum efforts, particularly when using the CUDA architecture.

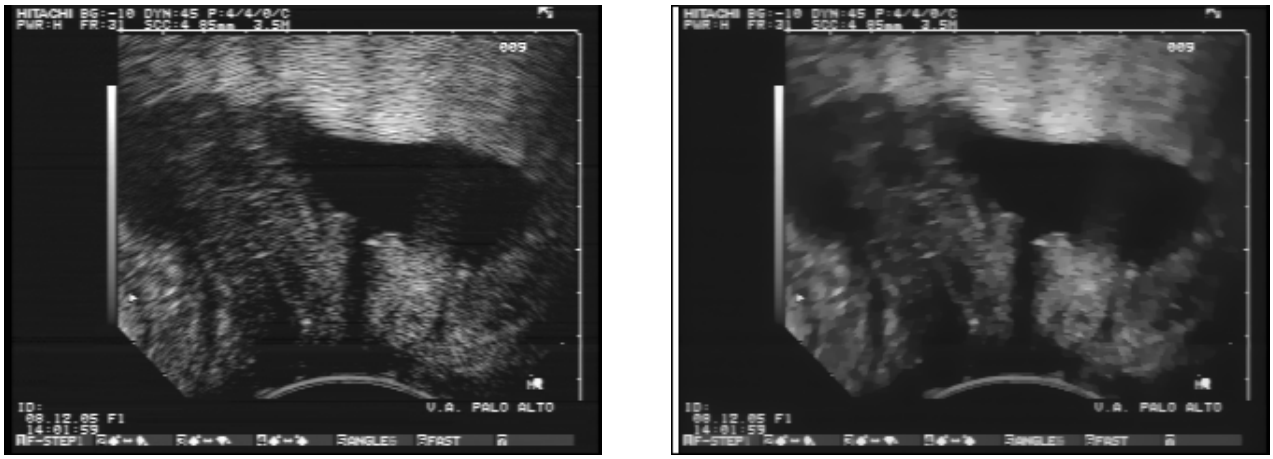


Figure 9: Original image and the image smoothed by the parallel implementation, respectively

6 Conclusions

The use of parallel computing techniques to fully explore the high-performance multiprocessor architecture is not new. However, the cost of the more traditional hardware for high-performance computing is not low; thus, more affordable alternatives such as GPU hardware should be considered.

The present work has described how to use the high-performance computing CUDA-based architecture as a computational infrastructure to accelerate an algorithm for noise image removal. The parallel GPU-based implementation developed was compared against the corresponding sequential CPU-based implementation in several experiments, and image quality metrics confirmed the similarity of the smoothing results achieved by each implementation. The parallelization of the image smoothing method based on a variational model using CUDA architecture reduced the runtime by up to 10.65 times in comparison with the CPU-based implementation.

The novel CUDA-based implementation developed to smoothing multiplicative noise by using an effective variational method seems to be a high-performance solution for applications with images susceptible to this type of noise, and which have high processing time constraints. Moreover, the proposed GPU-based parallelization approach has transparency, portability and scalability, thanks to the adopted SIMT model.

More and more complex methods and larger and larger data sets are used in the medical imaging domain that has high time constraints, which makes the use of the CUDA architecture extremely attractive as the study conducted here confirms. As a future works, we intend to extend the proposed CUDA-based implementation to enable it to perform in multi-GPUs, besides combining it with multi-thread (OpenMP) and multicomputer

(MPI) in order to achieve higher performances using heterogeneous parallel computing platforms.

7 Acknowledgments

The first author would like to thank the "Universidade do Estado de Mato Grosso" (UNEMAT), in Brazil, for the support given. The National Scientific and Technological Development Council (CNPq) partially supported this work through process 234360/2014-9 and Grant 2010/15691-0. Henrique Ferraz de Arruda thanks the Coordination for the Improvement of Higher Education Personnel (CAPES) for the financial support received. Authors gratefully acknowledge the funding of Project NORTE-01-0145-FEDER-000022 - SciTech - Science and Technology for Competitive and Sustainable Industries, cofinanced by "Programa Operacional Regional do Norte", (NORTE2020), through "Fundo Europeu de Desenvolvimento Regional" (FEDER).

References

[1] Z. Ma, J. M. R. S. Tavares, R. N. Jorge, and T. Mascarenhas. A review of algorithms for medical image segmentation and their applications to the female pelvic cavity. *Comput. Methods Biomech. Biomed. Eng.*, 13(2):235–246, 2010.

[2] H. Erives and G. J. Fitzgerald. Automated registration of hyperspectral images for precision agriculture. *Comput. Electron. Agric.*, 47(2):103–119, 2005.

[3] R. Arjona and I. Baturone. A hardware solution for real-time intelligent fingerprint acquisition. *J. Real Time Image Process.*, 9(1):95–109, 2014. ISSN 1861-8200.

[4] E. Todorovich, A. L. D. Pra, L. I. Passoni, M. Vazquez, E. Cozzolino, F. Ferrara, and G. Bioul. Real-time speckle image processing. *J. Real Time Image Process.*, 11(3): 535–545, 2013.

[5] I. Kunttu and L. Lepisto. Shape-based retrieval of industrial surface defects using angular radius Fourier descriptor. *IET Image Proc.*, 1(2):231–236, 2007.

[6] T. Mélange, M. Nachtegael, S. Schulte, and E. E. Kerre. A fuzzy filter for the removal of random impulse noise in image sequences. *Image Vis. Comput.*, 29(6): 407 – 419, 2011. ISSN 0262-8856.

[7] T. Aittokallio, J. Salmi, T. A. Nyman, and O. S. Nevalainen. Geometrical distortions in two-dimensional gels: applicable correction methods. *J. Chromatogr. B Anal. Technol. Biomed. Life Sci.*, 815(1-2):25–37, 2005.

[8] R. Jha, P. Biswas, and B. Chatterji. Contrast enhancement of dark images using stochastic resonance. *IET Image Process.*, 6(3):230–237, 2012.

[9] Y.-D. Wu, Y. Sun, H.-Y. Zhang, and S.-X. Sun. Variational PDE based image restoration using neural network. *IET Image Process.*, 1(1):85–93, 2007.

[10] M. Ezoji and K. Faez. Use of matrix polar decomposition for illumination-tolerant face recognition in discrete cosine transform domain. *IET Image Process.*, 5(1): 25–35, 2011.

[11] Z. Ma, R. N. M. Jorge, and J. M. R. S. Tavares. A shape guided C-V model to segment the levator ani muscle in axial magnetic resonance images. *Med. Eng. Phys.*, 32(7):766–774, 2010.

[12] F. P. M. Oliveira, T. C. Pataky, and J. M. R. S. Tavares. Registration of pedobarographic image data in the frequency domain. *Comput. Methods Biomech. Biomed. Eng.*, 3(6):731–740, 2010.

[13] L. Chen, M. Zhang, and Z. Xiong. Series-parallel pipeline architecture for high-resolution catadioptric panoramic unwrapping. *IET Image Process.*, 4(5): 403–412, 2010.

[14] V. Ponomaryov. Real-time 2D - 3D filtering using order statistics based algorithms. *J. Real Time Image Process.*, 1(3):173–194, 2007. ISSN 1861-8200.

[15] F. P. X. de Fontes, G. A. Barroso, P. Coupé, and P. Hellier. Real-time ultrasound image denoising. *J. Real Time Image Process.*, 6(1):15–22, 2011. ISSN 1861-8200.

[16] A. Merigot and A. Petrosino. Parallel processing for image and video processing: issues and challenges. *Parallel Comput.*, 34(12):694–699, September 2008.

[17] E. López-Rubio. Restoration of images corrupted by Gaussian and uniform impulsive noise. *Pattern Recogn.*, 43(5):1835–1846, 2010.

[18] Z. Jin and X. Yang. A variational model to remove the multiplicative noise in ultrasound images. *J. Math. Imaging Vis.*, 39(1):62–74, 2011.

[19] NVIDIA. *GPU Tutorial: Build environment, Debugging/Profiling, Fermi, Optimization/CUDA 3.1 and Fermi advice*. NVIDIA, 2010.

[20] N. Wilt. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley, Reading, 2013.

[21] D. Castano-Diez, D. Moser, A. Schoenegger, S. Pruggnaller, and A. S. Frangakis. Performance evaluation of image processing algorithms on the GPU. *J. Struct. Biol.*, 164(1):153 – 160, 2008. ISSN 1047-8477.

[22] G. A. Triantafyllidis, M. Varnuska, D. Sampson, D. Tzovaras, and M. G. Strintzis. An efficient algorithm for the enhancement of JPEG-coded images. *Comput. Graph.*, 27(4):529 – 534, 2003. ISSN 0097-8493.

[23] J. Ji. Robust approach to independent component analysis for SAR image analysis. *IET Image Process.*, 6(3):284–291, 2012.

[24] G. Aubert and J.-F. Aujol. A variational approach to removing multiplicative noise. *SIAM J. Appl. Math.*, 68(4):925–946, 2008.

[25] Y.-M. Huang, M. K. Ng, and Y.-W. Wen. A new total variation method for multiplicative noise removal. *SIAM J. Imaging Sci.*, 2(1):20–40, 2009.

[26] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *J. Phys. D*, 60(1-4):259–268, 1992.

[27] K. Krissian, R. Kikinis, C.-F. Westin, and K. Vosburgh. Speckle-constrained filtering of ultrasound images. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, number 1, 2005.

[28] F. P. M. Oliveira and J. M. R. S. Tavares. Medical image registration: a review. *Comput. Methods Biomech. Biomed Eng.*, 2014. 17(2):73-93.

[29] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4): 600–612, 2004.

[30] M. P. Eckert and A. P. Bradley. Perceptual quality metrics applied to still image compression. *Signal Process.*, 70(3):177–200, 1998.

[31] S. Winkler. Issues in vision modeling for perceptual video quality assessment. *Signal Process.*, 78(2):231–252, 1999.

[32] G. Ramponi, N. K. Strobel, S. K. Mitra, and T.-H. Yu. Nonlinear unsharp masking methods for image contrast enhancement. *J. Electron. Imaging*, 5(3):353–367, 1996.

[33] S. Hashemi, S. Kiani, N. Noroozi, and M. E. Moghaddam. An image contrast enhancement method based on genetic algorithm. *Pattern Recogn. Lett.*, 31(13): 1816–1824, 2010.

[34] O. Ghita and P. F. Whelan. A new GVF-based image enhancement formulation for use in the presence of mixed noise. *Pattern Recogn.*, 43(8):2646 – 2658, 2010. ISSN 0031-3203.

[35] Y. Shkvarko, A. C. Atoche, and D. Torres-Roman. Near real time enhancement of geospatial imagery via systolic implementation of neural network-adapted convex regularization techniques. *Pattern Recogn. Lett.*, 32(16):2197–2205, 2011.

[36] Q. Chen, Q. Sun, and D. Xia. Homogeneity similarity based image denoising. *Pattern Recogn*, 43(12):4089–4100, 2010.

[37] R. S. Alves and J. Tavares. Computer image registration techniques applied to nuclear medicine images. In J. M. R. da Silva Tavares and R. M. N. Jorge, editors, *Computational and Experimental Biomedical Sciences: Methods and Applications*, volume 21, pages 173–191. Springer, 2015.

[38] A. Nakhmani and A. Tannenbaum. A new distance measure based on generalized image normalized cross-correlation for robust video tracking and image recognition. *Pattern Recogn. Lett.*, 34(3):315–321, 2013.

[39] Z. Wang, A. C. Bovik, and L. Lu. Why is image quality assessment so difficult? In *ICASSP International Conference on Acoustics, Speech, and Signal Processing*, 2002.

[40] L. Zhang, W. Dong, D. Zhang, and G. Shi. Two-stage image denoising by principal component analysis with local pixel grouping. *Pattern Recogn.*, 43(4):1531–1549, 2010.

[41] W. M. Hwu. *GPU Computing GEMS*. Emerald ed. Morgan Kaufmann and NVIDIA, 2011. ISBN 978-0-12-384988-5.

[42] D. Kirk and W.-M. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, 2010. ISBN 978-0-12-381472-2.

[43] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. W. Kim. Design and performance evaluation of image processing algorithms on GPUs. *IEEE Trans. Parallel Distrib. Syst.*, 22(1):91–104, January 2011. ISSN 1045-9219.

[44] R. Farber. *CUDA Application Design and Development*. Elsevier, 2011.

[45] N. Kehtarnavaz and M. Gamadia. *Real-Time Image and Video Processing: From Research to Reality*. Morgan & Claypool Publishers, University of Texas at Dallas, USA, 1st edition, 2006.

[46] D. Levin, U. Aladl, G. Germano, and P. Slomka. Techniques for efficient, real-time, 3D visualization of multi-modality cardiac data using consumer graphics hardware. *Comput. Med. Imaging Graph.*, 29(6):463 – 475, 2005. ISSN 0895-6111.

Part B - Article 3

Detection of computationally-intensive functions in a medical image segmentation algorithm based on an active contour model

Abstract

Most image segmentation methods proposed in the literature are computationally expensive, especially when run on large medical datasets, and require powerful hardware to achieve image-based diagnosis in real time. This article describes our approach to detecting computationally-intensive functions in a competent medical image segmentation algorithm based on an active contour model. Profiling methods can assess an algorithm's performance concerning the overall cost of execution time, memory access, and performance bottlenecks. Our approach applies performance analysis techniques commonly available in traditional computing operating systems. Therefore, it does not require any new setup nor the development of new performance-measuring techniques, and thus ensures the shortest possible learning curve. This makes the approach potentially easy to be adopted and generalized by the researchers from the area of medical image processing and analysis. Hence, this article presents guidelines that can help researchers *a)* use profiling tools and *b)* detect and evaluate potential optimization snippets in medical image segmentation algorithms based on an active contour model by measuring overall performance bottlenecks. At a comparable cost in terms of execution time, both profiling tools tested in our experiments gather accurate data about the relationship between execution time and paths in the call graph. Additionally, the used call graph visualization provides users a quick graphical overview of the execution time of their codes, as well as throughput in memory accesses, and performance bottlenecks, which can significantly facilitate the performance analysis.

Keywords: Medical Image Processing and Analysis, Profiling Tools, Performance Analysis

1 Introduction

Image segmentation is one of the most critical operations performed on medical images, as it is responsible for identifying and delineating important regions within an image. In general, 3D vision, image registration, image classification, and interpretation tasks, just to name a few, depend on competent image segmentation steps in order to achieve the best results [1].

Active contour models (ACM), or "snakes" as they are widely known in the scientific community [2–5], offer an attractive approach to addressing contour detection problems. Furthermore, because of their application to fundamental medical image analysis problems, ACMs are capable of segmenting, matching, and tracking images of

anatomical structures by exploiting features such as the location, size, and shape of these structures. The Chan-Vese algorithm is a well-known image segmentation method based on active contours; it employs internal and external energy forces using graph theory to track a contour as it moves toward the structure of interest [6]. However, because this method requires complex calculations, there is a need to develop new optimization strategies in order to reduce the required runtime. Fortunately, the deployment of profiling methods has contributed effectively to identifying and evaluating portions of code responsible for excessive computational resources consumption [5, 7].

Profiling tools can accurately calculate the number of times a function is activated when an algorithm is operating as expected and can indicate timing information about the analyzed function [8]. Profiling information, at this level, is a useful tool in assisting algorithm's optimization based on collecting and measuring data related to the memory space, frequency, and duration of function calls, as well as time complexity of the algorithm under analysis. Several profiling tools, including `gprof` [8], `perf` [9], `tiptop` [10] and others [11–13], have been proposed to assist programmers in identifying performance bottlenecks when executing an algorithm on a CPU under a particular workload [9, 14].

The goal of this article was to identify time-consuming functions in the Chan-Vese ACM algorithm using profiling tools. Hence, the performance analysis based on profiling was adapted to effectively reduce the processing time of this algorithm and make it suitable for real-time diagnosis by simply exploiting all computational power commonly available in current personal computers. It should be noted that, here, the term "performance" refers to the efficiency of computing systems when executing algorithms, including the factors of throughput, latency, and availability.

This article is mainly focused on medical image processing and analysis applications and offers important guidelines to assist researchers in identifying time-consuming functions in their algorithms using profiling tools. The experimental results confirm that the obtained profiling information can identify most bottlenecks in the algorithm. This study also provides insight into why profiling information is valuable.

As far as the authors know, this is the first time that the adopted profiling tools were used to support the parallelization of a medical image segmentation algorithm. The findings contribute to better understandings in the image processing and analysis area. In this area, an increasingly higher number of images of high resolution need to be processed and analyzed rapidly in real clinical scenarios. Currently, computers with a multi-core processor are available in medical environments and, even though these computers are not always state-of-the-art models, their computational power is adequate to provide efficient image processing and analysis, if the used algorithms are efficiently implemented, which

makes paramount the insights and guidelines provided.

This article is organized by the following sections: Section 2 reviews the background of the problem tackled, section 3 presents methodologies commonly used in this research area, section 4 details our findings and discussion concerning the use of the profile information gathered about the medical image segmentation algorithm under study, and section 5 contains our conclusions and suggestions for future studies.

2 Background and Related Work

This section covers two topics: the segmentation of medical images, with focus on the Chan-Vese ACM algorithm, and the profiling method used to pinpoint bottlenecks that present excessive CPU consumption. Once found, these bottlenecks can be parallelized in order to achieve better computing performance by using multi-core processors.

2.1 Medical Image Segmentation

Segmentation is the partitioning of an image into its constituent homogeneous regions. This partitioning is commonly carried out on the desired feature(s), such as color, intensity, or texture [4, 15]. Medical image segmentation is crucial, for example, for the successful extraction of image features and their subsequent classification, and also for facilitating the visualization by performing the detection process more effectively. Briefly, medical image segmentation can have purposes such as image-based diagnosis and monitoring, and planning and navigation during surgery [3, 15].

Image segmentation methods have been applied in the partitioning of images acquired from a wide variety of objects, such as lungs [16], skin lesions [17, 18], and vessels [4, 19]. Different imaging modalities have been used to acquire clinically useful information about anatomical structures such as magnetic resonance, computed tomography, ultrasound, and others, and many image segmentation methods have been developed to segment the acquired images [4, 15, 16, 19].

Image segmentation techniques can be grouped into four categories: pixel-based, region-based, edge-based, and model-based [15]. Region- and pixel-based techniques are based on the concept of the discontinuity of pixel values, whereas these pixel values' similarity is the basis of the edge-based techniques. Region-based techniques rely on similar patterns in intensity values within a region of neighboring pixels; they include approaches such as thresholding, region growing, region splitting and merging. In the edge-based techniques, boundaries are detected based on abrupt changes in the intensity levels of an image; in these methods, the focus is on finding discontinuities - points,

lines and edges - concerning features such as color, intensity, or texture. In model-based techniques, image segmentation is framed as an statistical optimization problem [20].

By convention, image segmentation can be defined as the problem of finding a partition of a dataset $S_k \subset I$ into homogeneous regions, the union of which makes up the entire image I. Thus, the sets that perform a segmentation must satisfy:

$$I = \bigcup_{k=1}^{K} S_k, \tag{1}$$

where $S_k \bigcap S_j = \phi$ for $k \neq j$, and each S_k is connected. Ideally, a segmentation method finds those sets that correspond to distinct anatomical structures or regions of interest in the image.

The concept of active contours was introduced by Kass et al. [2]; it was widely accepted as a new approach for image segmentation. The main disadvantages of active contour models are related to the initial contour that must be close to the boundary of interest, and active contours have difficulties to progressing into boundary concavities. Based on that, the Chan-Vese algorithm was proposed by Chan and Vese [6] to deal with those limitations and shortcomings of the traditional snake model. The Chan-Vese algorithm is derived from a compilation of two other techniques: the level set method, used in edge detection through a topological change of the curves, and also the Mumford-Shah region-growing technique, applied in image segmentation.

The core computation in the Chan-Vese algorithm is the massive local window matching between input images, and this has proven to be a powerful and fast technique for both contour detection and region-based segmentation. In [6], the Chan-Vese algorithm was seen to have achieved significant results in the segmentation of different objects with various shapes and with inner contours.

In many cases, locations of boundaries are well detected and preserved by the Chan-Vese algorithm, even for objects whose boundaries are not defined by gradient or with very smooth boundaries. Internal forces are computed within the curve to keep it smooth throughout the deformation. External forces are usually derived from the input images to drive the curve towards the desired features of interest [3]. An active contour model moves according to its dynamic equations and performs successive minimization iterations of a given energy associated with the curve. The Chan-Vese ACM is based on variational methods and each successive iteration is updated with the preceding curve points. The energy functional of the Chan-Vese model is defined in terms of the level set

function $\phi(x,y)$ as follows:

$$\begin{aligned} F(c_1,c_2,f) = \mu \cdot \int_o d_e(f(x,y))|\nabla f(x,y)|dxdy \\ +\lambda_1 \int_o |u_0(x,y)-c_1|^2 H_e(f(x,y))|dxdy \\ +\lambda_2 \int_o |u_0(x,y)-c_2|^2 (1-H_e(f(x,y)))|dxdy \end{aligned} \tag{2}$$

where μ, λ_1 and λ_2 are positive constants used to modulate the contribution of each term; f is any variable curve, and the constants c_1, c_2, depending on f, are average intensity inside and outside a zero level set, respectively. This minimization problem is solved using the level set method which replaces the unknown curve f by the level set function $\phi(x,y)$, considering that $\phi(x,y) > 0$ if the point (x,y) is inside f, $\phi(x,y) < 0$ if (x,y) is outside (x,y) and $\phi(x,y) = 0$, if (x,y) is on f. $H_\varepsilon(z)$ and δ_ε are the regularized approximation of Heaviside function $H(z)$ and Dirac delta function $\delta(z)$ as follows:

$$H(z) = \begin{cases} 1 & \text{if } z \geqslant 0, \\ 0 & \text{if } z < 0. \end{cases} \quad and \quad d(z) = \frac{d}{dz}H(z). \tag{3}$$

Equation 2 is performed by successive iterations, and at each iteration, the curve is updated point by point; hence, by analyzing the neighborhood of each point, it is possible to calculate the energy involved, move the curve towards image features, and approach the object boundary [2, 20]. The Chan-Vese method was already validated by various numerical results such as, for example, in [6]. The Chan-Vese algorithm is presented in Algorithm 2:

2.2 Profiling Method

This section reviews the employment of profiling in measuring the time required for each function in a computer algorithm. Profiling is a method commonly used to discern the behavior of an algorithm and measure its performance through the process of collecting information during execution.

Program profiling is typically used to measure the use of the instruction set to identify and assess portions of code presenting excessive CPU consumption; in addition, it is used to locate both memory allocation, usage, or leaks, cache performance, execution time, or even energy consumption [13]. Profiling methods include instrumented, event-based, statistical, and simulation [7–9].

Algorithm 2: Chan-Vese segmentation algorithm.

Input: Image $I(x,y)$
Preprocessing;
    Compute feature map (Input Image I);
    Compute gradient map $G = \nabla G_\sigma \oplus I$;
    Normalize G;
    Compute regional information-based normalized feature map F;
Initialize φ;
for $n = 1, 2, \ldots, N_{max}$ **do**
    Search the 3 x 3 neighborhood;
    Compute c_1 and c_2 as the region averages;
    Evolve φ with one semi-implicit timestep;
    if $\|\varphi^{n+1}$ - $\varphi^n\|_2 / |\Omega| \leq tol$ **then**
        stop;
    end
end
Update the contour information;

Performance analysis based on profiling usually follows these main steps: instrumentation or modification of the algorithm to produce performance data, measuring notable aspects of execution, which generates the performance data, analysis, and visualization of the performance data [12], as shown in Fig. 1.

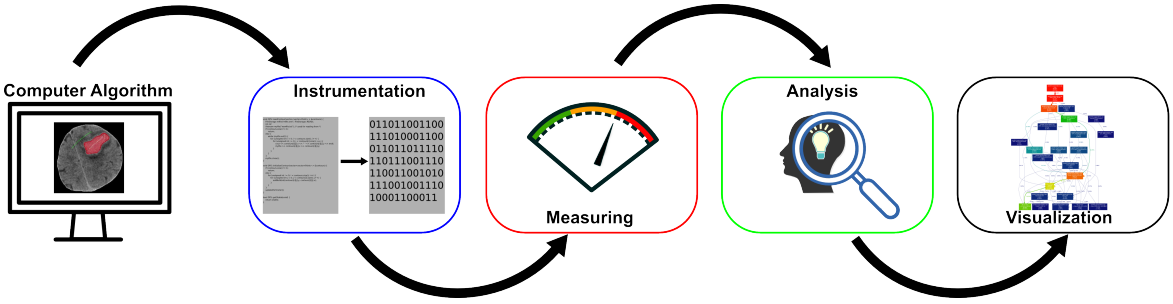


Figure 1: Diagram of the Profiling Method. Each part of the diagram shown is described in the text.

2.2.1 Instrumentation

Instrumentation is the process of incorporating measurement code into an algorithm at compile time, resulting in a much more precise measurement of execution times. This procedure adds to the object file a detailed listing of the running statistics and links the executable with standard libraries that have profiling information enabled. However, it requires the availability of the source code and the compiler [5, 12].

When using a profiling method, one should note that: first and foremost, the algorithm behavior should be modified as little as possible; the monitoring of the runtime behavior of algorithms involves instrumenting the binaries to record desired events; and the system event data, in essence, keep track of the interaction between programs and the hardware.

2.2.2 Measuring

Gathering profile data is the second step of the profiling method; it is responsible for monitoring hardware interrupts, operating system calls and performance counters [8]. After performance data from one or more executions have been recorded, information relating to functions is extracted and then stored in output files. In general, gathering profiling data does not interfere with the execution of the algorithm.

Performance counters are available in most modern processors; they enable count hardware performance events such as clock per cycles, floating-point operations, cache misses. In summary, performance counters are incremented when either comparison or arithmetical instructions are issued.

2.2.3 Data Analysis

The resulting binary leads to an output file named `perf.data` for the `perf` profiler, and another one named `gmon.out` for `gprof`, both of which contain the execution profile. Data is analyzed to extract performance statistics, for instance, the number of times each function is called and the time spent on each function. The profilers also record the arc in the call graph that is responsible for activating that function [8, 9, 12]. Information like the returning address for a function is used for identifying the source of the arc, call named *caller*, and the destination of the arc named *callee* [11].

When it comes to finding the most costly function in an algorithm, it is critical to collect the arcs of the dynamic call graph traversed by the execution of the algorithm. Hence, in post-processing this data, it is possible to visualize the call graph graphically and to represent the measures collected from the algorithm execution.

2.2.4 Visualization

In the last step, the gathered data can be visualized. Call stack walking is a technique that shows the inner workings of any algorithm even without access to the source code. This technique can show what functions are called and the CPU usage time for each function. The `gprof` and `perf` provide dynamic call graph information for all instrumented code snippets. A call graph is binary and sometimes is treated as multi-graph, instead of as relations - relation over functions, or procedures, defined in an algorithm [8, 9]. The

edges represent all the calls between the functions executed by the algorithm together with the call frequency. The nodes show the individual functions in the executable.

2.3 Related Work

With the emergence of multi-core processor architectures, one can no longer avoid parallelizing applications. Besides, while writing parallel algorithms from scratch has always been considered a difficult task, parallelizing legacy algorithms written by someone else, today a common scenario in the medical image processing and analysis area, is even harder [21]. On the other hand, several studies have been proposed to address the segmentation of medical images using high-performance computing [21–23]; however, it is not common for medical image processing and analysis developers make use of profiling methods to detect costly function in their algorithms.

An updated overview of image processing and analysis methods accelerated by high-performance computing architectures is given by Gulo et al. [24]. Many authors deploy approaches of image segmentation based on thresholding [25, 26], clustering [27] and deformable models [28], on a PC-cluster [27, 29, 30], using graphics processing unites (GPUs) [21, 25, 31–35] or multi-core processors [27, 35].

Daggett and Greenshields [29] and Yeh and Fu [30] designed a parallel algorithm using a PC-cluster to segment magnetic resonance (MR) images in order to reduce the inter-process communication overhead. This parallel algorithm was based on the virtual shared memory technique, which enables processes to communicate by directly sharing data as though it existed in a global shared memory space. This approach was designed using the Message Passing Interface (MPI) programming model and the Single Program, Multiple Data Stream (SPMD) data decomposition model. Examples of application include automating the clinical diagnosis of schizophrenia and multiple sclerosis. In the Gabriel et al. [27] approach, they used a multi-core processor and a PC-cluster to compare the speed-up, communication overhead, different memory systems, and different number of used threads. The multi-core architecture achieved the highest speed-ups, which were up to 11x faster compared to the PC-cluster.

The performance of GPUs was exploited to accelerate image segmentation algorithms, such as level set-based segmentation [25, 33] and Bias Field Correction Fuzzy C-Mean [21]. However, the expensive computation required by the algorithms demanded optimization strategies in order to reduce the run-time; hence, Lamas-Rodríguez et al. [33] aimed to divide the active domain of the input images into fixed-size tiles and therefore, intensively use shared memory space, resulting in a low latency close to that of the register space. Balla-Arabé and Gao [25] designed a selective entropy-based energy

functional method, robust against noise, and new selective entropy external forces for the Lattice Boltzmann method (LBM). However, neither Lamas-Rodríguez et al. [33] nor Balla-Arabé and Gao [25] approaches achieved volume image segmentation in real time. Hence, the authors identified a need for future studies to extend their approach to a GPU cluster environment. In the approach of Aitali et al. [21], the GPU implementation achieved real time in segmenting volume images.

Zhuge et al. [31, 32] took advantage of the CUDA architecture, mainly by supporting atomic read/write operations in the GPU global memory, in order to develop a semi-automatic segmentation method based on the Fuzzy Connected technique. Shi et al. [34] proposed an automatic image segmentation method for medical images based on a Pulse Coupling Neural Network combined with the 2D Tsallis entropy, resulting in stronger adaptability and high image segmentation precision. The results with this GPU-based approach was in real time using ray tracing.

In the Saran et al. [35] approach, a rigid mutual information registration of magnetic resonance venography (MRV)/magnetic resonance angiography (MRA) images was used to increase vessel segmentation accuracy in MR images. The unfavorable effects of Rician noise and Radio-frequency (RF) inhomogeneity in MR, MRA, and MRV images during vessel segmentation are removed by applying a subtraction scheme. In this scheme, the cost function and choice of the minimization method are executed simultaneously using multi-core and GPU.

3 Material and Methods

As described in Section 2.1, the usual image segmentation algorithm consists of multiple steps, including general tasks such as image reading and setting up the segmentation parameters. On the other hand, opportunities to optimize their implementations can be identified by recognizing parallelization options, by using profiling tools [5, 36], see Section 2.2.

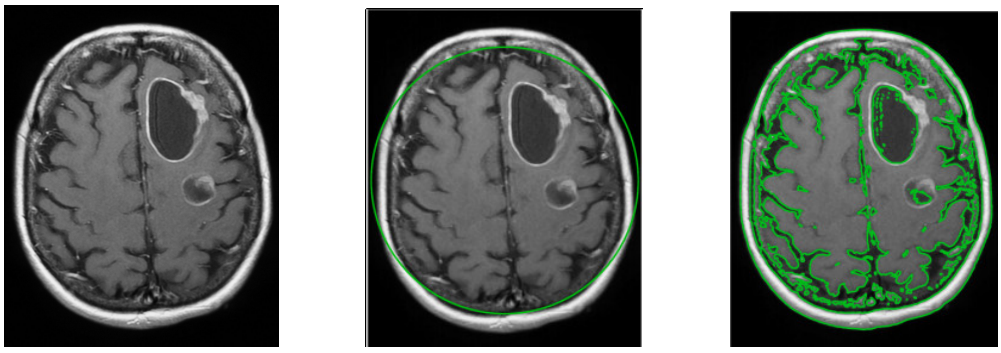
3.1 Experimental Setup

The used test infrastructure included a desktop computer equipped with Linux Debian 8 operating system, GNU gcc/g++ compiler version 4.9.2, `gprof` 2.25, `perf` 3.16.7-ckt20,

`gprof2dot` [10], and `dot` [11] 2.38, 16 GB of RAM (DDR3-1600 Mhz), and an Intel(R) Core(TM) i7-4790 3.60 GHz processor. This processor has four physical cores, with each one being capable of running two logical threads simultaneously.

3.2 Dataset

This study used Multiple Sclerosis (MS) images selected from the MS Longitudinal Challenge Data Set repository [37]. These images are readily available for research purposes. Thirteen images were chosen from the initial dataset to validate the studied image segmentation method. The randomly selected images were built and preprocessed in the same manner, with the data acquired using a 3.0 Tesla MR imaging scanner (Philips Medical System, Best, The Netherlands) according to the following parameters: T_1-weighted $(T_1 - w)$ magnetization prepared rapid gradient echo (MPRAGE) with TR=10.3 ms, TE=6 ms, flip angle=8°, and 0.82x0.82x1.17 mm^3 voxel size; a double spin echo (DSE) which produces PD-w and $T_2 - w$ images with TR=4177 ms, TE$_1$=12.31 ms, TE$_2$=80 ms, and 0.82×0.82×2.2 mm^3 voxel size; and a $T_2 - w$ fluid attenuated inversion recovery (FLAIR) with TI=835 ms, TE=68 ms, and 0.82x0.82x2.2 mm^3 voxel size [37]. Figure 2 shows an example of a segmentation obtained by the algorithm under analysis.



(a) (b) (c)

Figure 2: Segmentation of a MR brain image:(a) Original image, (b) Segmentation initializing, (c) Segmentation obtained using the Chan-Vese algorithm.

[10] A `gprof2dot` is an open source script written in Python used to convert output from a range of profiles into a dot graph. This script can be downloaded for free at `https://github.com/jrfonseca/gprof2dot`.

[11] A `dot` is a Graphviz feature for producing hierarchical drawings of directed graphs. Graphviz is an open source visualization software for representing structural information such as diagrams of abstract graphs. More information is available at `http://graphviz.org`.

3.3 Segmentation Results Evaluation

Dice Similarity Coefficient (DSC) is a statistical validation metric commonly used to evaluate the performance of both the reproducibility of ground truth segmentations and the spatial overlap accuracy of automated probabilistic fractional segmentations. The DSC value is a simple and useful summary measure of spatial overlap, which can be applied to studies of reproducibility and accuracy in image segmentation [38]. The DSC value ranges from 0 (zero) indicating no spatial overlap between two segmentation results to 1 (one) indicating complete overlap. Hence, the DSC measures the spatial overlap between two segmentations, X, and Y, and is defined as:

$$DSC = \frac{2|X \bigcap Y|}{|X| + |Y|}, \tag{4}$$

where $|X|$ and $|Y|$ are the number of pixels in X and Y, respectively, with X being the area of the segmentation obtained by the segmentation algorithm, Y the area of the ground truth segmentation and $Y \bigcap Y$ the overlapping area of the two segmentations.

3.4 Performance Evaluation

In order to measure the performance of the Chan-Vese algorithm, we focused on the runtime required by each function in this algorithm, using the profiling tools: `gprof` and `perf`. We selected `gprof` and `perf` tools because they can combine three profiling methods: instrumented, event-based, and statistical. On one hand, `gprof` is commonly considered easy to use and portable, although it is limited in scope; it is designed to produce a detailed call graph identifying the functions responsible for calling other functions and how many times they were called. Besides, `gprof` provides information about the number of calls to each function, lists the percentage of time spent in a function, and computes the amount of time needed to execute that function. On the other hand, `perf` makes use of statistical sampling to collect profile data thereby generating an interruption at regular time intervals. All process running on the CPU is identified by `perf`, which then captures all relevant information such as the program counter, CPU core number; it then writes all of this data to an output file called `perf.data`.

The Chan-Vese single thread-based algorithm was compiled with parameters that create a working executable: *a)*`-fno-omit-frame-pointer`, this enables frame pointer analysis; *b)*`-g`, used to generate symbol information and in turn enables source code analysis; and *c)*`-pg`, used to compile and link the source code with profiling information enabled - the monitor function `mcount` is inserted before each function call.

The compiler parameter `-pg` generates the monitoring function named *mcount*, which is immediately called by each profiled function, and *mcount* returns the address were is recorded. This address falls inside the profiled function that is the destination of an arc in the call graph. The monitoring function also identifies the source of the arc. Arcs represent invocations in the same function and are named cycles. When a child function is a member of a cycle, the time shown is the appropriate fraction of the time for the whole cycle. Self-recursive routines have their calls broken down into calls from the outside and self-recursive calls; thus, only the outside calls affect the propagation of time. It is important to point out that the algorithm calls `libc-2.19` are related to the `C` runtime library, and that is not uncommon to spend significant amounts of time in a runtime library and not in the algorithm code itself.

4 Results and Discussion

In this section, we provide results of experiments aimed at getting useful profiling information, accumulating data producing statistically meaningful observations, and reducing measurement errors of the Chan-Vese algorithm. For this purpose, each test image was segmented by the Chan-Vese algorithm. However, it was not the goal of the present article to assess the accuracy of the used segmentation algorithm, rather we focused on measuring the performance of functions on the Chan-Vese's implementation and on its speed-up for the multi-thread implementation.

4.1 Algorithm Evaluation

A quantitative evaluation was performed to analyze the obtained segmentation results; i.e., the ground truth of the segmented regions were used to confirm whether each lesion presented in the 13 images were correctly segmented or not. Hence, the segmentation results obtained by our implementation were compared against the ground truths using Dice Similarity Coefficient, Table 1.

4.2 Runtime Evaluation

For performance evaluation, we measured the running time in seconds using a C++ function for all the reported experiments. Each experiment was executed fifty times for each image; then, the mean and standard deviation values of the time required to segment each input image were calculated, including the time spent to load the data into the main

Table 1: Direct comparison of ground truth and algorithm-based segmentation results for 13 images via the Dice Similarity Coefficient (DSC).

Image	Dimension	DSC
# 1	256x256x35	0.93062
# 2	256x256x120	0.94457
# 3	256x256x70	0.94104
# 4	256x256x70	0.95521
# 5	256x256x70	0.94667
# 6	256x256x120	0.95533
# 7	256x256x70	0.95167
# 8	256x256x70	0.94509
# 9	256x256x70	0.96565
# 10	256x256x120	0.95143
# 11	256x256x70	0.94293
# 12	256x256x70	0.95363
# 13	256x256x120	0.96453

system memory until the end of the segmentation process, when the resultant image was produced, Table 2.

Table 2: Means and standard deviations of the runtime (in seconds) required by the sequential-based Chan-Vese algorithm implementation.

Image	Dimension	Runtime
# 1	256x256x35	14.086114 $\pm$ 0.01345
# 2	256x256x120	15.946149 $\pm$ 0.03271
# 3	256x256x70	14.280519 $\pm$ 0.01961
# 4	256x256x70	13.486183 $\pm$ 0.01808
# 5	256x256x70	15.202701 $\pm$ 0.03133
# 6	256x256x120	12.112178 $\pm$ 0.07894
# 7	256x256x70	15.680396 $\pm$ 0.32751
# 8	256x256x70	15.297073 $\pm$ 0.41024
# 9	256x256x70	13.940298 $\pm$ 0.14520
# 10	256x256x120	14.173801 $\pm$ 0.14520
# 11	256x256x70	15.082609 $\pm$ 0.14520
# 12	256x256x70	15.830090 $\pm$ 0.14520
# 13	256x256x120	13.926398 $\pm$ 0.20677

4.3 Performance analysis

Gathering profile data was then the next step performed, including collecting data while monitoring hardware interrupts, operating system calls and performance counters. Profiling tools periodically interrupt the kernel of the operating system to record a new sample and then save the samples that are stored in the ring buffer, generating overhead. `perf` mitigates sampling overhead thereby enforcing sampling buffer locality when `perf` creates one instance of the event on each CPU; then, the events are effectively measured when the thread is executed on that CPU. All the samples are aggregated into a single

output file once all profiles have been run. In the experiments conducted for this study, the sampling mode in `perf` was used in order to trace the Chan-Vese algorithm events in real time, `perf` generates output files larger than the ones resulting from the `gprof` profiler. The size of the output files from the `gprof` profiler was 7.9 KB (for experiments with 2, 4, and 8 threads) and 16 KB (for experiments with 1 thread), instead of dozens of megabytes when obtained from the `perf` profile, as indicate in Table 3. This massive difference in profiling data sizes is because a `gprof` output file stores a histogram of algorithm counter samples and the arc table; `perf` depends on the frequency - the rate of 4000 samples per second - at which events are recorded, resulting in higher overhead and larger output files.

Table 3: File sizes, indicated in kilo (KB) and megabytes (MB), generated by `perf` according to the images dimension and the OpenMP-based implementation with different number of threads.

Image	**Dimension**	**Number of threads**			
		1 Thread	2 Threads	4 Threads	8 Threads
# 1	256x256x35	938 KB	1 MB	2.1 MB	2.8 MB
# 2	256x256x120	2.1 MB	5 MB	6.8 MB	8 MB
# 3	256x256x70	1.5 MB	4.5 MB	2 MB	3.4 MB
# 4	256x256x70	1.46 MB	4.5 MB	2.2 MB	4.1 MB
# 5	256x256x70	1.65 MB	4.5 MB	2.4 MB	4.3 MB
# 6	256x256x120	2.1 MB	5 MB	6.8 MB	8 MB
# 7	256x256x70	1.36 MB	4.5 MB	6.7 MB	7.6 MB
# 8	256x256x70	1.37 MB	4.5 MB	7.8 MB	7.5 MB
# 9	256x256x70	1.36 MB	4.5 MB	6.9 MB	7.8 MB
# 10	256x256x120	2.1 MB	5 MB	6.9 MB	7.8 MB
# 11	256x256x70	1.6 MB	4.5 MB	6.9 MB	7.8 MB
# 12	256x256x70	1.6 MB	4.5 MB	6.9 MB	7.8 MB
# 13	256x256x120	2.1 MB	5 MB	6.8 MB	8 MB

The collected data was analyzed to extract performance statistics and also to record the arc in the call graph responsible for activating each implemented function. Call graph represents time-consuming functions and the number of times the functions were invoked. By analyzing the call graph sample from the segmentation of image #1, the call graph shown in Fig. 3 was generated, which includes the time required for each function from its descendants, and the number of times each function was called.

The call graph normally displays the children as well as the parents of each function in the graph, including the higher level functions that consume large portions of the total execution time in the functions that they call. In the context of this study, children mean functions that are called by another (parent) function. In Fig. 3, five items are indicated by numbered circles: item 1 indicates the name of the caller function; item 2 concerns the percentage of algorithm runtime accounted for the function and its children; item 3 represents time with different meanings depending on whether it is the primary function

for that section, the function's parent or child functions. In the first case, the time indicates the time spent on that function during the execution of the algorithm. In the second case, it shows the amount of the first self-time function being propagated for that parent, based on the percentage of calls to the primary function made by the parent. Finally, for child functions, it represents the amount of that child function self-time propagated for the primary function based on the percentage of calls made to that function by the primary function; item 4 is related to the number of times that function was called; and finally, item 5 regards the percentage of total function time propagated for each child function.

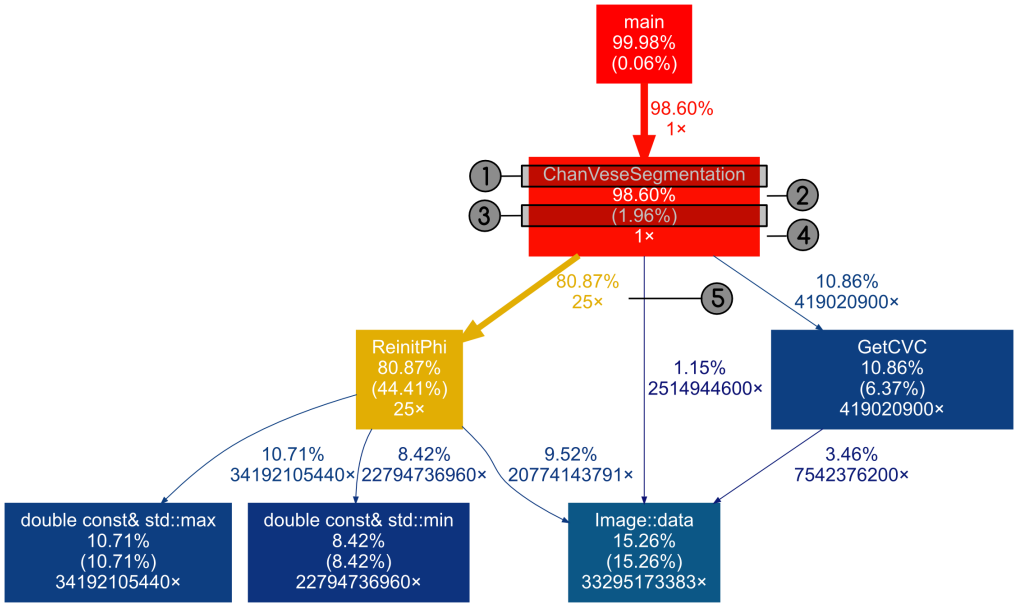


Figure 3: Call graph generated by `perf` representing the most often functions called by the Chan-Vese algorithm.

The call graph helps focus the analysis on the relevant parts of the algorithm execution, making the experiments easier to understand. The `main` function called the `ChanVeseSegmentation` function, and this one, called the functions `GetCVC`, `ReinitPhi`, `Image::data`, `min`, and `max`. Function `ReinitPhi` is responsible for locally computing the signed distance function to its zero level set, and was identified by our method as the most called and the one that required the most part of the running time: around 80% of the total running time (12.90 seconds), see Algorithm 2, from line 6 to line 12. `GetCVC` function computes the coefficients needed in the Chan-Vese algorithm for the level set function. `Image:data` function is used to assign the point to minimal energy neighborhood, see Algorithm 2, line 2; the auxiliary functions `min` and `max` are used in the minimization of the functional with respect to c_1, c_2, and f (line 9 of Algorithm 2).

Fig. 4 represents the list of the most used functions in the studied image segmentation algorithm. The first measurement shown in Fig. 4 reflects the time spent on each

function based on counter events. All these times were obtained by running the Chan-Vese algorithm fifty times and calculating the average of the time elapsed, as reported by the profiling tool. In all cases, the execution times for different runs of the implementation were extremely consistent. Functions `ReinitPhi` and `GetCVC` were the most frequent in the run stack; they were responsible for occupying the processor for 23.50 seconds (in terms of exclusive time), and this means 98.60% of the full runtime. In fact, these functions iterated a hundred times, which makes them attractive parallelization targets. Based on the large amount of work it performs, the algorithm under study exhibits a high degree of instruction parallelism, since every element on the input image can be computed independently.

Because of the high number of iterations and computations per iteration, the segmentation algorithm under analysis is considered to be computationally-intensive so that the most promising target was identified among the suggestions generated by the profiling tools, i.e. the function that, by iterating over the lines of the input image and solving Equation 2, represents the most demanded computation, and therefore most of the execution time is consumed.

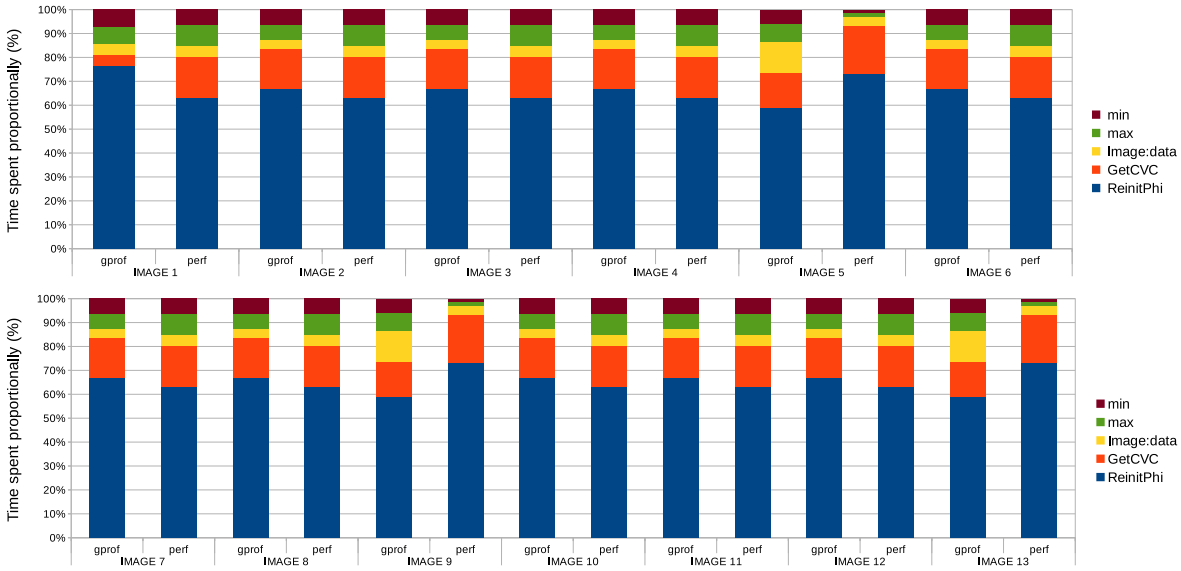


Figure 4: Most time-consuming functions detected by the profiling tools `perf` and `gprof`.

4.4 Effected of the number of used cores

Finally, we discuss the effect of using a different number of physical cores on the performance of the multi-threaded Chan-Vese segmentation algorithm. For a fixed number of cores, we used an equal number of threads for the execution; mainly, one thread

for each core. The costly function `ReinitPhi`, was implemented using OpenMP. All the experiments previously performed were repeated and then compared using different degrees of parallelism: 1, 2, 4 and 8 threads. As shown in Fig. 5, the experiments using the parallel OpenMP-based implementation revealed a reduction in the runtime of the segmentation algorithm relatively to the single-thread implementation.

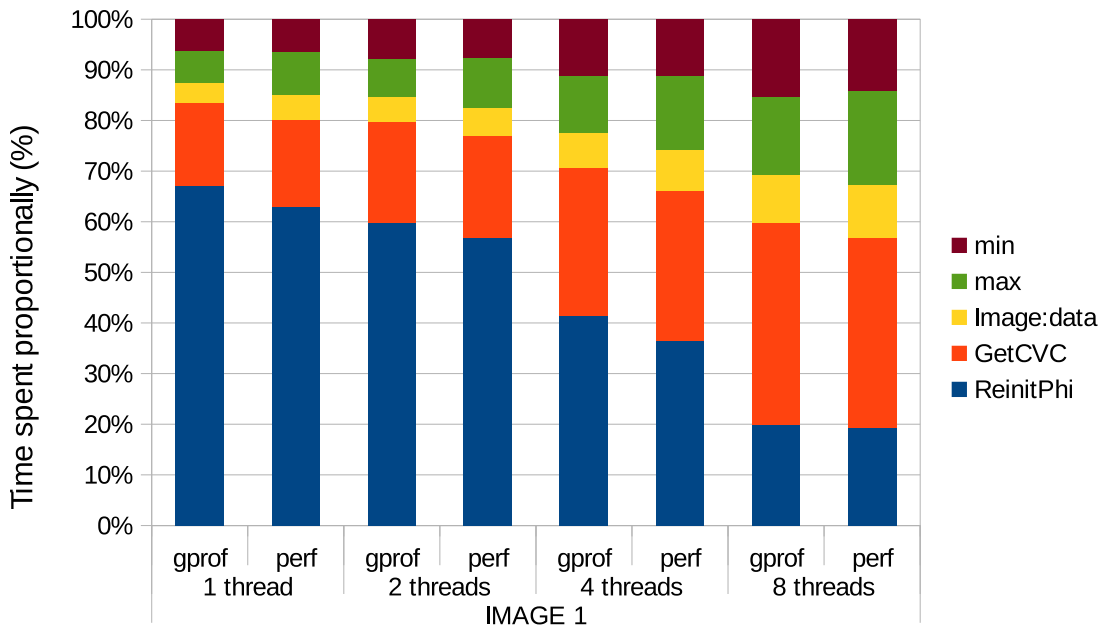


Figure 5: Time-consuming functions detected by the profiling tools `perf` and `gprof` using OpenMP-based implementation of the Chan-Vese algorithm.

When considering images with dimensions bigger than 768x576 pixels, a speed-up of the OpenMP-based implementation was evident. Fig. 6 suggests that the performance scales almost exponentially, being the processing time of the parallel implementation about 7 times faster than the single thread-based implementation. Hence, the performance gain of the parallel OpenMP-based implementation confirms the high processing capacity available in multi-core processors. For a fixed number of cores, we used an equal number of threads for the execution: one thread for each core. It is clear from Fig. 6 that for each image size, the execution time decreased as we increased the number of cores. Therefore, our findings confirm that computational parallelization assisted by profiling tools can increase the application performance and facilitate implementation efforts.

5 Conclusion and Future Works

The present work has described how to use profiling tools to detect and evaluate performance bottleneck snippets in an image segmentation algorithm based an the active

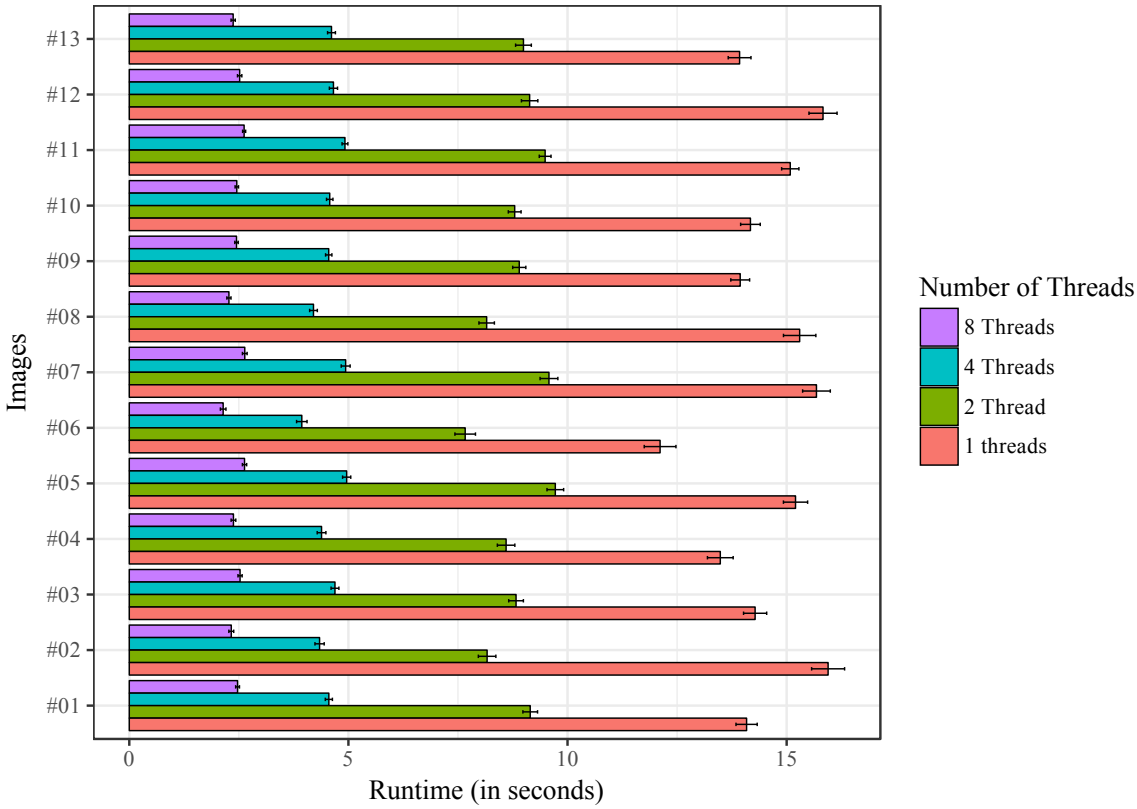


Figure 6: Means and standard deviations of runtime spent for running the OpenMP-based implementation of the Chan-Vese algorithm.

contour. The developed parallel OpenMP-based implementation was compared against the corresponding single thread-based implementation in several experiments. The parallelization of the costly function of the Chan-Vese algorithm reduced the runtime by up to 7 times compared to the single thread-based implementation.

The novel profiling model applied to medical image processing and analysis seems to be an elegant solution for applications that have high processing time constraints. The profiling method provided a detailed profile that is combined with the source level information to identify and evaluate performance bottleneck snippets in the Chan-Vese algorithm. Not only does the combined approach detect the available parallelism targets, but it also substantially reduces the overall time needed to parallelize the sequential application.

As our findings confirm, parallel programming can provide substantial acceleration in processing speed. In our study, the processing time decreased in all cases, as the number of threads increased. The speed-up of the execution process relies on many factors, including compiler optimizations, runtime support, data layout, operating system noise, workload balancing and so on. Additionally, it may depend on the regions needing

to be merged in the structure of the graph of the input image. In a parallel loop, even if the threads complete their process, except for one, they will have to wait for this thread. In these cases, the operating system might have executed another task on that thread in the meantime. Nevertheless, the time saved through using parallelization is remarkable and promising.

In future studies, we plan to further optimize the time-consuming functions already detected and described in this article by using heterogeneous parallel computing platforms based on GPUs. Program parallelization assisted by profiling tools would not only increase the maximum application performance and reduce the required manual implementation efforts, but also provide a psychological incentive for developers to adopt this methodology.

6 Acknowledgments

The first author gratefully thanks for the support given: the Universidade do Estado de Mato Grosso (UNEMAT) of Brazil, and the National Council for Scientific and Technological Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq), process 234306/2014-9 grants under reference #2010/15691-0.

References

[1] J. Duan, Z. Pan, X. Yin, W. Wei, and G. Wang. Some fast projection methods based on Chan-Vese model for image segmentation. *EURASIP Journal on Image and Video Processing*, 2014(1):7, Jan 2014. ISSN 1687-5281. doi: 10.1186/1687-5281-2014-7. URL https://doi.org/10.1186/1687-5281-2014-7.

[2] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1:321, 1988. ISSN 0920-5691. doi: https://doi.org/10.1007/BF00133570.

[3] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91 – 108, 1996. ISSN 1361-8415. doi: http://dx.doi.org/10.1016/S1361-8415(96)80007-7. URL http://www.sciencedirect.com/science/article/pii/S1361841596800077.

[4] D. L. Pham, C. Xu, and J. L. Prince. A survey of current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–337, August 2000. doi: 10.1146/annurev.bioeng.2.1.315.

[5] Z. Li, R. Atre, Z. Huda, A. Jannesari, and F. Wolf. Unveiling parallelization opportunities in sequential programs. *Journal of Systems and Software*, 117: 282 – 295, 2016. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2016.03.045. URL `http://www.sciencedirect.com/science/article/pii/S016412121630005X`.

[6] T. Chan and L. Vese. An active contour model without edges. In M. Nielsen, P. Johansen, O. F. Olsen, and J. Weickert, editors, *Scale-Space Theories in Computer Vision: Second International Conference*, volume 1682 of *Lecture Notes in Computer Science*, pages 141–151, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48236-9. doi: 10.1007/3-540-48236-9_13. URL `https://doi.org/10.1007/3-540-48236-9_13`.

[7] S. Rul, H. Vandierendonck, and K. D. Bosschere. A profile-based tool for finding pipeline parallelism in sequential programs. *Parallel Computing*, 36(9): 531 – 551, 2010. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2010.05.006. URL `http://www.sciencedirect.com/science/article/pii/S0167819110000840`.

[8] S. L. Graham, P. B. Kessler, and M. K. McKusick. gprof: A call graph execution profiler. *ACM SIGPLAN Notes*, 39(4):49–57, April 2004. ISSN 0362-1340. doi: 10.1145/989393.989401. URL `http://doi.acm.org/10.1145/989393.989401`.

[9] M. Dimakopoulou, S. Eranian, N. Koziris, and N. Bambos. Reliable and efficient performance monitoring in Linux. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-4673-8815-3. URL `http://dl.acm.org/citation.cfm?id=3014904.3014950`.

[10] E. Rohou and INRIA. Tiptop: Hardware performance counters for the masses. In *2012 41st International Conference on Parallel Processing Workshops*. IEEE, 2012. doi: 10.1109/ICPPW.2012.58.

[11] M. Schulz and B. R. de Supinski. Practical differential profiling. In *Euro-Par 2007 Parallel Processing*, pages 97–106. Lecture Notes in Computer Science, Springer,

2007. ISBN 978-3-540-74466-5. doi: 10.1007/978-3-540-74466-5_12. URL http://dx.doi.org/10.1007/978-3-540-74466-5_12.

[12] J. M. Spivey. Fast, accurate call graph profiling. *Softw. Pract. Exper.*, 34(3):249–264, March 2004. ISSN 0038-0644. doi: 10.1002/spe.562. URL http://dx.doi.org/10.1002/spe.562.

[13] T. Ball and J. R. Larus. Optimally profiling and tracing programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1319–1360, July 1994. ISSN 0164-0925. doi: 10.1145/183432.183527. URL http://doi.acm.org/10.1145/183432.183527.

[14] S. Shende. Profiling and tracing in Linux. In *Proc. Second Extreme Linux Workshop, USENIX Annual Technical Conference*, pages 26–30, 1999.

[15] S. Masood, M. Sharif, A. Masood, M. Yasmin, and M. Raza. A survey on medical image segmentation. *Current Medical Imaging Reviews*, 1:3–14, 2015. doi: 10.2174/1573405611011504231034 41.

[16] P. P. R. Filho, P. C. Cortez, A. C. d. S. Barros, V. H. C. Albuquerque, and J. M. R. S. Tavares. Novel and powerful 3D adaptive cristp active contour method applied in the segmentation of CT lung images. *Medical Image Analysis*, 35:503–516, 2017. ISSN 1361-8415. doi: 10.1016/j.media.2016.09.002.

[17] R. B. Oliveira, M. E. Filho, Z. Ma, J. P. Papa, A. S. Pereira, and J. M. R. S. Tavares. Computational methods for the image segmentation of pigmented skin lesions: A review. *Computer Methods and Programs in Biomedicine*, 131:127–141, 2016. ISSN 0169-2607. doi: 10.1016/j.cmpb.2016.03.032.

[18] R. B. Oliveira, N. Marranghello, A. S. Pereira, and J. M. R. S. Tavares. A computational approach for detecting pigmented skin lesions in macroscopic images. *Expert Systems with Applications*, 61:53 – 63, 2016. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2016.05.017. URL http://www.sciencedirect.com/science/article/pii/S0957417416302354.

[19] D. S. Jodas, A. S. Pereira, and J. M. R. S. Tavares. Lumen segmentation in magnetic resonance images of the carotid artery. *Computers in Biology and Medicine*, 79:233 – 242, 2016. ISSN 0010-4825. doi: http://dx.doi.org/10.1016/j.compbiomed.2016.10.021. URL http://www.sciencedirect.com/science/article/pii/S0010482516302827.

[20] H. Lu, Y. Li, Y. Wang, S. Serikawa, B. Chen, and J. Chang. Active contour model for image segmentation: A review. In *International Conference on Industrial Applications Engineering*, pages 104–111, 2013. doi: 10.12792/iciae2013.022.

[21] N. Aitali, B. Cherradi, A. E. Abbassi, and O. Bouattane. Parallel implementation of bias field correction fuzzy C-Means algorithm for image segmentation. In *(IJACSA) International Journal of Advanced Computer Science and Applications*, volume 7(3), pages 375–383, 2016. doi: 10.14569/IJACSA.2016.070352.

[22] M. Jeon, M. Alexander, and N. Pizzi. Parallel image segmentation with level set methods. In *Proceedings on the 5th IASTED International Conference on Visualization, Imaging, and Image Processing*, pages 394–399, Bonidorm, Spain, 2005.

[23] D. Bader, J. Jaja, D. Harwood, and L. S. Davis. Parallel algorithms for image enhancement and segmentation by region growing with an experimental study. In *Proceedings of International Conference on Parallel Processing*. IEEE, 1996. ISBN 0-8186-7255-2. doi: 10.1109/IPPS.1996.508089.

[24] C. A. S. J. Gulo, A. C. Sementille, and J. M. R. S. Tavares. Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review. *Journal of Real-Time Image Processing*, Nov 2017. ISSN 1861-8219. doi: 10.1007/s11554-017-0734-z. URL `https://doi.org/10.1007/s11554-017-0734-z`.

[25] S. Balla-Arabé and X. Gao. Geometric active curve for selective entropy optimization. *Neurocomputing*, 139:65–76, 2014. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2013.09.058.

[26] P. Saiviroonporn, A. Robatino, J. Zahajszky, R. Kikinis, and F. Jolesz. Real-time interactive three-dimensional segmentation. *Academic Radiology*, 5(1):49–56, JAN 1998. ISSN 1076-6332. doi: 10.1016/S1076-6332(98)80011-1.

[27] E. Gabriel, V. Venkatesan, and S. Shah. Towards high performance cell segmentation in multispectral fine needle aspiration cytology of thyroid lesions. *Computer Methods and Programs in Biomedicine*, 98(3):231–240, 2010. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2009.07.008.

[28] M. Salomon, F. Heitz, G.-R. Perrin, and J.-P. Armspach. A massively parallel approach to deformable matching of 3D medical images via stochastic differential

equations. *Parallel Computing*, 31(1):45–71, 2005. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2004.12.003.

[29] T. Daggett and I. Greenshields. A cluster computer system for the analysis and classification of massively large biomedical image data. *Computers in Biology and Medicine*, 28(1):47–60, 1998. doi: 10.1016/S0010-4825(97)00032-2. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031807067&doi=10.1016%2fS0010-4825%2897%2900032-2&partnerID=40&md5=a00bfbbb29bf230c3fefe9ce1e7f0f78.

[30] J.-Y. Yeh and J. Fu. Parallel adaptive simulated annealing for computer-aided measurement in functional MRI analysis. *Expert Systems with Applications*, 33 (3):706–715, 2007. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2006.06.018.

[31] Y. Zhuge, Y. Cao, J. K. Udupa, and R. W. Miller. Parallel fuzzy connected image segmentation on GPU. *Medical Physics*, 38(7):4365–4371, JUL 2011. ISSN 0094-2405. doi: 10.1118/1.3599725.

[32] Y. Zhuge, K. C. Ciesielski, J. K. Udupa, and R. W. Miller. GPU-based relative fuzzy connectedness image segmentation. *Medical Physics*, 40(1), JAN 2013. ISSN 0094-2405. doi: 10.1118/1.4769418.

[33] J. Lamas-Rodríguez, D. B. Heras, F. Argüello, D. Kainmueller, S. Zachow, and M. Bóo. GPU-accelerated level-set segmentation. *Journal of Real-Time Image Processing*, 12(1):15–29, Jun 2016. ISSN 1861-8219. doi: 10.1007/s11554-013-0378-6. URL https://doi.org/10.1007/s11554-013-0378-6.

[34] W. Shi, Y. Li, Y. Miao, and Y. Hu. Research on the key technology of image guided surgery. *Przeglad Elektrotechniczny*, 88(3B):29–33, 2012. ISSN 0033-2097.

[35] A. N. Saran, F. Nar, and M. Saran. Vessel segmentation in MRI using a variational image subtraction approach. *Journal of Electrical Engineering and Computer Sciences*, 22(2):499–516, 2014. ISSN 1300-0632. doi: 10.3906/elk-1206-18.

[36] S. Prema and R. Jehadeesan. Analysis of parallelization techniques and tools. *International Journal of Information and Computation Technology*, 3(5):471–478, 2013. ISSN 0974-2239.

[37] A. Carass, S. Roy, A. Jog, J. L. Cuzzocreo, E. Magrath, A. Gherman, J. Button, J. Nguyen, F. Prados, C. H. Sudre, M. J. Cardoso, N. Cawley, O. Ciccarelli, C. A. Wheeler-Kingshott, S. Ourselin, L. Catanese, H. Deshpande, P. Maurel, O. Commowick, C. Barillot, X. Tomas-Fernandez, S. K. Warfield, S. Vaidya, A. Chunduru, R. Muthuganapathy, G. Krishnamurthi, A. Jesson, T. Arbel, O. Maier, H. Handels, L. O. Iheme, D. Unay, S. Jain, D. M. Sima, D. Smeets, M. Ghafoorian, B. Platel, A. Birenbaum, H. Greenspan, P.-L. Bazin, P. A. Calabresi, C. M. Crainiceanu, L. M. Ellingsen, D. S. Reich, J. L. Prince, and D. L. Pham. Longitudinal multiple sclerosis lesion segmentation: Resource and challenge. *NeuroImage*, 148:77 – 102, 2017. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2016.12.064. URL `http://www.sciencedirect.com/science/article/pii/S1053811916307819`.

[38] K. H. Zou, S. K. Warfield, A. Bharatha, C. M. Tempany, M. R. Kaus, S. J. Haker, W. M. Wells, F. A. Jolesz, and R. Kikinis. Statistical validation of image segmentation quality based on a spatial overlap index. *Academic Radiology*, 11(2): 178–189, 2004. ISSN 1076-6332. doi: https://doi.org/10.1016/S1076-6332(03)00671-8. URL `http://www.sciencedirect.com/science/article/pii/S1076633203006718`.

Part B - Article 4

Optimizing a medical image registration algorithm based on profiling data towards real-time performing

Abstract

A considerable number of algorithms have been developed to perform rigid and nonrigid registration, which is a task commonly conducted in medical image analysis. Particularly, the free-form deformation algorithm is frequently used to carry out nonrigid registration; however, it is a very compute-intensive algorithm. Herein, we describe our approach to identifying potential parallelism parts of this algorithm and exploiting their parallel implementations using profiling data. Our approach assesses the performance of the algorithm under study by applying performance analysis techniques commonly available in traditional computer operating systems. Hence, this article presents guidelines to support researchers working on medical image processing and analysis to achieve real-time nonrigid image registration applications using common computing systems. According to our experimental findings, significant speedups can be accomplished by parallelizing sequential snippets, i.e. code regions that are executed more than once. Based on the application programming interface OpenMP of the costly functions previously identified in the studied free-form deformation algorithm, the developed parallelization decreased the runtime by up to seven times relatively to the single thread-based implementation. In conclusion, this study confirms that one can easily detect and evaluate potential optimization snippets, in addition to throughput in memory accesses, based on the call graph visualization and detected performance bottlenecks.

Keywords: Medical image processing and analysis, profiling tools, performance analysis, nonrigid image registration

1 Introduction

The analysis of medical images plays a significant role in medicine. Image registration is an important and widely used technique in this context. Nowadays, patients are imaged on a routine basis using different imaging systems. Patients are also monitored over time to assess disease progression or response to therapy. However, to be able to study physiological and/or structural changes over time, or to combine complementary information that different imaging systems produce, it is necessary to perform the registration of the acquired images [1].

Image registration is a computational task that determines the spatial correspondence between two images of the same object acquired at different angles or time or using different image modalities, or under different acquisition conditions [2–4]. In general, an image registration method can be decomposed into three parts: a transformation model, a

similarity measure and an optimization process [4, 5]. Transformation models delineate the transformation that can be used to represent the underlying correspondences: rigid models, describe simple linear mappings such as translations, rotations, scalings and shears; on the other hand, nonrigid transformation models can represent mappings that are much more complex since local deformations are also taken into account usually resulting in very time-consuming processes [5, 6]

Nonrigid image registration is an extensive research field, encompassing many applications. It includes several specific algorithms; among others, there are the ones based on mutual information [7, 8], elastic transformations model [9], multi-resolution [10], and similarity measures [6]. However, many issues related to the high required computational efforts are commonly encountered when nonrigid image registration algorithms are used. Therefore, nonrigid image registration is well-known in the literature as one of the most time-consuming tasks in medical image analysis [11, 12].

Beginning with the development of multi-core processor architecture, several solutions have been proposed to deliver nonrigid image registration algorithms on multi-core CPUs [13–15]. Although multi-core architecture was developed to improve the performance of applications exploiting parallelism, writing parallel algorithms from scratch is a very complex and demanding task. Furthermore, parallelizing legacy algorithms written by someone else is even more challenging [16–18].

The deployment of a profiling method can contribute effectively to the identification and evaluation of portions of code responsible for excessive computational resources consumption [17, 18]. For example, a profiling tool can count the exact number of times a function is activated when the algorithm under analysis is running, and display timing information about that function [19]. At this level, profiling is a helpful approach in program optimization based on gathering and calculating data regarding memory space, frequency, duration of function calls, and time complexity of an algorithm. Many profiling tools, like `gprof` [19], `perf` [20], `tiptop` [21] and others [22–24], have been proposed to help programmers identify performance bottlenecks during the execution of algorithms on CPU under a particular workload [17, 20, 23].

With this work, we aimed to identify high time-consuming functions in one of the most popular image registration algorithms used: the Free-Form Deformation (FFD) algorithm [11, 12], using profiling tools. Therefore, performance analysis based on profiling data was used to effectively decrease the processing time of the algorithm and adapt it to be suitable for real-time diagnosis by exploiting all the computational resources typically available in modern personal computers. Here, the term “performance” refers to the efficiency of computer operating systems while executing algorithms, including factors of throughput, latency, and availability.

Therefore, throughout this article, we provide guidelines and methods that can support researchers of medical image processing and analysis in identifying very time-consuming functions in their algorithms using profiling tools. The experimental findings show that this profiling information can identify the majority of the bottlenecks in a real C implemented algorithm. This study also provides insight into why profiling data is useful, particularly to optimizing a nonrigid image registration algorithm towards real-time application.

To the best of our knowledge, this is the first time that the adopted profiling tools were used as support in parallelization of a nonrigid image registration algorithm. Our findings are therefore, highly pertinent to the image processing and analysis area, mainly for the medical imaging community. In this area, medical images of more and more higher resolution must be processed and analyzed as quickly as possible in real clinical scenarios. Additionally, computers with multi-cores are available in medical environments and, even though these computers are not always the most up-to-date ones, their computational power is still sufficient for efficient tasks of image processing and analysis. Therefore, the insights to be presented are timely and demanded for researchers developing efficient algorithms of medical image processing and analysis.

This article is organized as follows: Section 2 introduces the related background; then, it is presented the profiling method used to identify snippets that present excessive CPU consumption; afterwards, lists reviews relevant in the literature on methods that speedup the computation of nonrigid image registration algorithms. The material and methods used to speedup the studied algorithm of nonrigid image registration, including the profiling tools, regarding tasks such as measuring algorithm performance, gathering data to be analyzed, and building the visualization of the performance analysis, are addressed in Section 3. Our main findings and the discussion of our experience with the use of profile data in order to optimize the computation of the image registration algorithm are presented in Section 4. Section 5 provides the conclusion of this study.

2 Background and related works

This section introduces the topic of medical image registration and the used profiling tools. Next, we review related research regarding medical image registration algorithms that have been speedup by high-performance computing techniques.

2.1 Medical image registration

Image registration is the process of aligning images of the same object obtained at different times and or from different viewpoints, using different or similar imaging modalities/conditions [8, 16, 25]. This process geometrically combines two images, which are usually known as the reference and sensed images. Image registration is a critical step in image analysis tasks where the desired information can be gathered from the combination of various data sources as in image fusion, change detection, and multichannel image restoration, just to name a few [14, 26]. Here, we focus on nonrigid registration, where the changes between the images are due to usual global rotations, translations and scaling, but also due to complex local variations. Medical image registration is also commonly used to follow up information on patient anatomy along different time points, where one must account for deformation of the anatomy itself due to, for example, the patient's breathing or normal anatomical changes [9, 14].

As already aforementioned, a considerable number of image registration methods have been developed both to obtain the combination, i.e. the fusion, of data acquired by different clinically useful imaging modalities through mutual co-registration, for example, or to register one image to other images to understand how patient anatomy has changed over time [14, 15]. In general, the majority of the rigid image registration methods consist of four steps: feature detection, feature matching, transform model estimation, and image resampling and transformation [14, 25]. On the other hand, the nonrigid registration methods commonly search for the optimal transformation parameters that maximise a similarity measure. All these steps are well documented in the literature [14, 15, 25, 27], and details are omitted for brevity.

Medical image nonrigid registration should establish a correspondence measure between a reference image, I_r, and sensed image, I_s, using a parameter transformation, $T_t(\cdot)$, of image geometry in line with a similarity function, $\rho(\cdot)$, to specify the registration performance. When I_s has a higher dimension than I_r, projection operators P_r and P_s can be used to reduce I_s dimensionality. Then, the nonrigid image registration problem can be expressed via maximizing the similarity measure function [26]:

$$T_t^*(\cdot) = \arg_{T_t(\cdot)} \max \rho(P_r(I_r), P_s(T_t(I_s))). \tag{1}$$

An FFD model comprises a powerful tool for deforming an image volume using cubic B-splines. This technique is applied, for example, in deformation analysis in brain images, by deforming an object by adjusting an underlying mesh of control points, creating the 3D shape of the object, and a smooth and C^2 continuous transformation [12]. To define a

spline-based FFD, we denote the domain of the image volume as $\Omega = \{(x,y,z)|0 \leq x < X, 0 \leq y < Y, 0 \leq z < Z\}$. Let the parameters of the transformation and the amount of deformation Φ denote a $n_x \times n_y \times n_z$ mesh of control points $\phi_{i,j,k}$ with uniform spacing δ. Thus, ϕ can be formed regarding a low resolution mesh for modeling global nonrigid deformations, and high resolution mesh for more accurately modeling local deformations of the control points mesh [11, 12]. Thus, the FFD can be written as the 3D tensor product of 1D cubic B-splines, which can be expressed as:

$$T_{local^{(x,y,z)}} = \sum_{l=0}^{3}\sum_{m=0}^{3}\sum_{n=0}^{3} B_l(u)B_m(v)B_n(w)\phi_{i+l,j+m,k+n}, \tag{2}$$

where $i = \lfloor x/n_x \rfloor - 1$, $j = \lfloor y/n_y \rfloor - 1$, $k = \lfloor z/n_z \rfloor - 1$, $u = x/n_x - \lfloor x/n_x \rfloor$, $v = y/n_y - \lfloor y/n_y \rfloor$, and $w = z/n_z - \lfloor z/n_z \rfloor$. B_l represents the l-th basis function of the B-spline [11, 12]:

$$\begin{aligned} B_0(u) &= (1-u)^3/6, \\ B_1(u) &= (3u^3 - 6u^2 + 4)/6, \\ B_2(u) &= (-3u^3 + 3u^2 + 3u + 1/6, \\ B_3(u) &= u^3/6. \end{aligned} \tag{3}$$

Considering $B_l(u) = 0$ for $l < 0$ and $l > 3$, the derivative terms are nonzero only in the neighborhood of a given point. Therefore, the optimization of the objective function using gradient descent can be efficiently achieved [11, 12]. However, the FFD algorithm is computationally intensive, requiring considerably time to compute, particularly when dealing with images of huge dimension, which is very common in several possible applications [14]. For example, the parallel computation of the human brain deformation is a new field of exploration, and it can be more efficiently studied through processing large amounts of high-resolution images concurrently [3]. Also, the used conjugate gradient descent algorithm can optimize all control points and interpolate the whole image at each iteration [11]. However, the computation of the similarity measure and of the geometric transformation are the computational bottlenecks of the nonrigid registration algorithms. Thus, researchers should focus more attention on developing more effective parallelization techniques for these computations.

2.2 Profiling methods

This section introduces the use of profiling methods for measuring the time needed by each function in a computer algorithm. Profiling is a well-known tool that evaluates

algorithms performance through gathering data during their execution, particularly, in order to assist programmers in identifying performance bottlenecks.

Algorithm profiling is commonly used to understand an algorithm's performance and to assess the use of an instruction set in order to identify and evaluate portions of code requiring excessive processor consumption; likewise, it is used to identify both memory allocation, usage or leaks, cache performance, execution time, or even energy consumption [22]. There are different profiling approaches such as instrumented, event-based, statistical, and simulation [18–20].

Performance analysis based on profiling usually involves four different steps: instrumentation or modification of the algorithm under study to generate performance data, measurement of noteworthy aspects of execution, which generates the performance data, analysis and visualization of the performance data [24], Fig. 1.

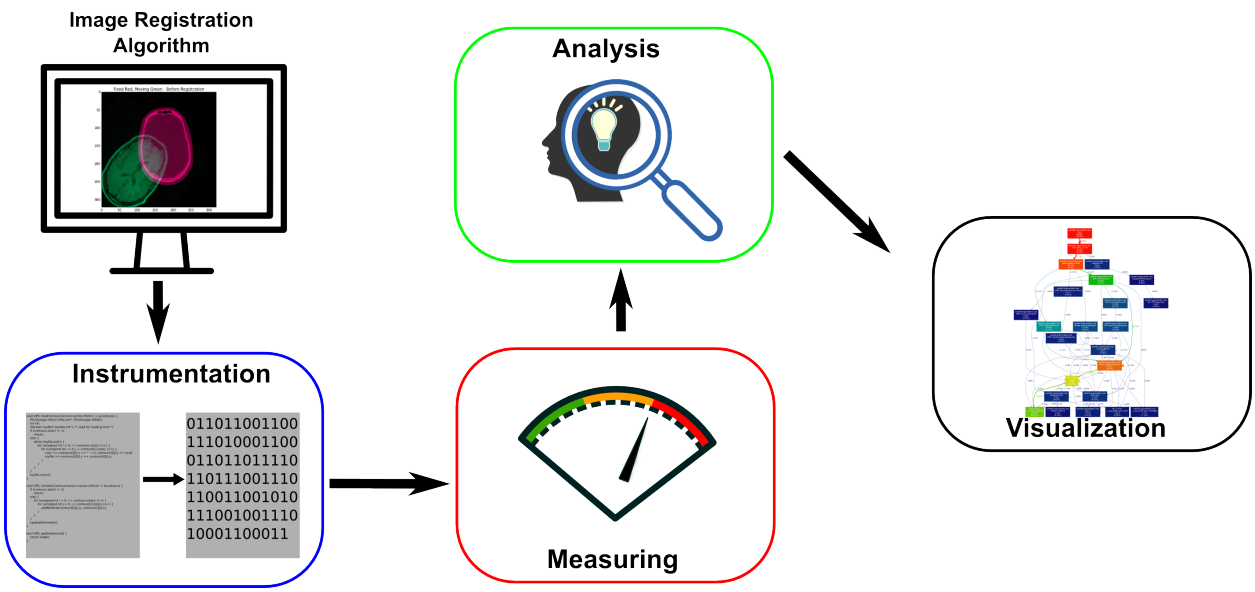


Figure 1: Diagram of the Profiling Method. The function of each stage of the diagram shown is described in the text.

2.2.1 Instrumentation

Instrument an algorithm implementation requires the availability of the source code and the compiler, thus at compile time, a detailed listing of the running statistics are added to the object file, and the executable is linked to standard libraries that have profiling information enabled. At this point, the instrumentation incorporates measurement code into the implementation, resulting in an accurate assessment of running times [17, 24, 28]. All instrumentation processes are developed in order to determine how the algorithm's behavior should be modified. Monitoring runtime behavior of algorithms involves

aggregating information on the base of the number of executions of every basic-block, instrumenting binaries to trace various type of events such as *free* and *malloc* and similar function utilities.

2.2.2 Measuring

Gathering profile data is the second step of the profiling method, consisting of gathering the following information during algorithm execution: the approximate time spent in each function; the number of times a function is invoked; a list of the caller functions invoking a given function; a list of the descendant functions that a given function invokes; and an estimate of the cumulative time spent in the descendant functions invoked by a given function [19]. By post-processing of this information from one or more executions, information relating to functions is gathered and then stored in output files. Therefore, a dynamic call graph for the execution is created [18, 23]. In general, gathering profiling data does not interfere with the execution of the algorithm under analysis [23, 24].

2.2.3 Data Analysis

In the third step of the profiling method, the related binary is produced, and the output data is available for extraction. The output files are named `perf.data` for the `perf` profiler, and `gmon.out` for `gprof`, respectively, and each file contain the execution profile. These profilers analyze the data and extract performance statistics, besides recording the arc in the call graph for activating each function [19, 20, 24].

At this stage, the profiler determines the most costly functions and collects the arcs of the dynamic call graph traversed by the execution of the algorithm under evaluation. Thus, enables to visualize the call graph graphically and to represent the measures collected from the algorithm execution. Information like the returning address for a function call named *caller* that is used for identifying the source of the arc and the destination, which is named *callee* [23].

2.2.4 Visualization

In the final step, the gather profiling data is presented by incorporating the call graph of the algorithm under analysis. Call stack walking is a technique that identifies calling relationships between functions in an implementation. In this technique, every call relationship that occurs is represented in the graph with the CPU usage time for each function call.

Both `gprof` and `perf` tools provide dynamic call graph information for all instrumented code snippets. A call graph is binary and sometimes is treated as a

multi-graph, instead of as relations-relation over functions, or procedures, defined in an algorithm implementation [19, 20]. Each edge (f, g) shows that function f invokes function g; and the nodes show the individual functions in the executable.

2.3 Related works

In high-performance computing, parallel computing has been applied to highly complex problems such as computing huge workload and data, and intensive critical analysis. Sequential algorithm implementations are frequently re-coded in order to decompose the algorithms or the data into smaller portions. These portions are commonly named as tasks, and are distributed to be executed in many- or multi-cores, simultaneously [29, 30]. Throughout all this procedure, the tasks of communication and coordination are performed based on memory usage by different computer processing units [30].

The growing popularity and use of multi-core processor architectures in medical imaging applications have been documented [14, 15, 27] in the overview of multi-core computing. Multi-core CPUs were designed to increase the performance of applications exploiting parallelism; however, writing parallel implementations from scratch is a very complex and demanded challenge. Besides, parallelizing legacy implementations written by someone else is even harder [14].

Based on literature reviews presented by [26, 31, 32], it is clear that it is uncommon for the medical image processing and analysis developers to use tools to detect computationally costly functions in their algorithms; however, several studies have been proposed to address performance issues in image registration algorithms using high-performance computing [13, 15, 33]. For example, Shackleford et al. [14] performed a comprehensive survey of nonrigid registration algorithms that are suitable for use in modern multi-core architectures.

Due to their high parallelism, image registration tasks are computationally costly. Therefore, multi-core computing with their high-performance parallel processing power provides excellent opportunities for speeding up these tasks.

Computationally intensive, Mutual Information-based (MI-based) algorithms have been successfully employed in parallel architectures such as clusters [34], Graphic Processing Unit (GPU) [27, 33], multi-core Cell Broadband Engine Architecture (CBEA) [35], and Field-Programmable Gate Array (FPGA) [7], reducing their runtime and making them suitable for routine clinical use. For example, MI-based algorithms have been used to correct the misalignment of tissue in computed tomography (CT), positron emission tomography (PET) and magnetic resonance (MR) images, achieving accuracy comparable to one achieved by clinical experts.

Rohlfing and Maurer [3] and Christensen [34] exploited the use of shared-memory multiprocessor computer architectures as well as data and task partition parallel programming models. Rehman et al. [2] developed a parallel approach of nonrigid registration by regarding it as an Optimal Mass Transport problem. Lapeer et al. [36] presented a point-based registration method, integrating a Radial Basis Function (RBF) as a smoothing function and sought to mimic the interacting deformation of biological tissues. Mafi and Sirouspour [37] exploited a GPU-based computational platform for real-time analysis of soft object deformation.

Ellingwood et al. [16] developed a new computation- and memory-efficient Diffeomorphic Multi-Level B-Spline Transform Composite method on GPU for the nonrigid mass-preserving registration of CT volumetric images. The Sum of Squared Tissue Volume Difference (SSTVD) was adopted as the similarity criterion to preserve the computed tissue volume. A cubic B-Spline-based Free-Form Deformation transformation model was used to capture the nonrigid deformation of objects like human lungs. The experiments used lung CT images, indicating an increase of speed of 112 times relative to the single-threaded CPU version, and of 11 times compared to the 12-threaded version when considering the average time per iteration using the GPU implementation.

3 Material and Methods

As described in Section 2.1, the nonrigid image registration algorithm under study involves of transforming different sets of data into one coordinate system. To accelerate the FFD algorithm, the transformation of the floating image using the splines and an interpolation function, evaluation of an objective function, besides the optimization of this function, are taken into account. Acceleration possibilities for the optimization step were identified by recognizing parallelization options, through the use of profiling tools [17, 18, 24].

3.1 Environment settings

The used test infrastructure included a desktop computer, with a Linux Debian 8 operating system, GNU gcc/g++ compiler version 4.9.2, `gprof` 2.25, `perf` 3.16.7-ckt20, `gprof2dot` [12], and `dot` [13] 2.38, 16 GB of RAM (DDR3-1600 Mhz), and an Intel(R)

[12] `gprof2dot` is an open source script written in Python used to convert the output from a range of profiles into a dot graph. This script can be downloaded for free at `https://github.com/jrfonseca/gprof2dot`.

[13] `dot` is a Graphviz feature for producing hierarchical drawings of directed graphs. Graphviz is an open source visualization software for representing structural information such as diagrams of abstract graphs.

Core(TM) i7-4790 3.60 GHz processor. This processor has four physical cores, and two logical threads can be run simultaneously in each core.

This study used Multiple Sclerosis (MS) images, which were collected from the MS Longitudinal Challenge Data Set repository [38]. The images are freely distributed for research purposes. Thirteen images were randomly selected from the original dataset to validate the nonrigid image registration results. The selected images were scanned and preprocessed in the same manner, with the data acquired using a 3.0 Tesla MR imaging scanner (Philips Medical System, Best, The Netherlands) according to the following parameters: T_1-weighted $(T_1 - w)$ magnetization prepared rapid gradient echo (MPRAGE) with TR=10.3 ms, TE=6 ms, flip angle=8°, and 0.82x0.82x1.17 mm^3 voxel size; a double spin echo (DSE), which produces PD-w and $T_2 - w$ images with TR=4177 ms, TE_1=12.31 ms, TE_2=80 ms, and 0.82×0.82×2.2 mm^3 voxel size; and a $T_2 - w$ fluid-attenuated inversion recovery (FLAIR) with TI=835 ms, TE=68 ms, and 0.82x0.82x2.2 mm^3 voxel size [38].

3.2 Registration evaluation

Dice Similarity Coefficient (DSC) is a simple and useful statistical validation metric commonly used to evaluate the performance of both registration reproducibility and spatial overlap accuracy against to registration ground truths [11]. The DSC value rates the overlap of two masks between 0 (zero) and 1 (one), where 1 (one) indicates a perfect overlap and 0 (zero) none. Therefore, DSC assesses the spatial overlap between the registration result (M_m) and the corresponding registration ground truth (M_p) as:

$$DSC = \frac{2\|M_m \bigcap M_p\|}{\|M_p\| + \|M_p\|}, \tag{4}$$

where $\|M_m\|$ and $\|M_p\|$ are the number of pixels, or voxels in 3D, in M_m and M_p, respectively, M_m is the area, or volume in 3D, of the registration obtained by the automated algorithm, M_p the area, or volume, of the ground truth registration and $M_p \bigcap M_p$ the overlapping area, or volume, of the two images under comparison.

More information is available at `http://graphviz.org`.

3.3 Performance evaluation

In order to estimate the speedup of the studied Free-Form Deformation algorithm, Amdahl's law of speedup can be used:

$$Speedup_{enhanced} = \frac{1}{(1-f)+\frac{f}{S}}, \tag{5}$$

where $Speedup_{enhanced}$ is the overall speedup of the algorithm, f the execution time of a function eligible for optimization, and S the expected speedup of this function. The key idea of this formula is to determine functions in an implementation that are more time-consuming and can be speedup using optimization. Such a function (or a part of it) is often referred to as a bottleneck. To gain significant overall speedup, the value of f should be high [29, 30, 39]. Once the bottlenecks are identified, optimizations are postulated to help improve their performance. These optimizations should then be individually verified to ensure that they result in measurable improvements.

The performance of the FFD algorithm under study was then improved regarding the bottlenecks identified through using the profiling tools `gprof` and `perf`. These tools were selected because they combine profiling methods based on instrumentation, event-based, and statistics. Both tools consist of two parts: a runtime routine, a call to which is inserted by the compilers at the beginning of every function compiled with profiling parameters; and a post-processing version of the algorithm under analysis that aggregates and presents the data. We compiled the Free-Form Deformation single thread-based algorithm implementation with the following parameters: (`-fno-omit-frame-pointer`) in order to enable the frame pointer analysis; (`-g`) for generating symbol information, which in turn enabled source code analysis; and the parameter `-pg`, which is used for inserting the monitor function `mcount` before each function call.

The monitor function *mcount* records the function address and identifies the source of the cycles based on the addresses generated inside the profiled function. When a child function is a member of a cycle, the time shown is the appropriate fraction of the time for the whole cycle. Self-recursive routines have their calls broken down into calls from the outside and self-recursive calls; thus, only the outside calls affect the time propagation.

`gprof` is considered easy to use and portable, although it is limited in scope; it is designed to produce a detailed call graph identifying the functions responsible for calling other functions and the number of times their functions were called. Furthermore, `gprof` lists the percentage of time spent in a function and computes the amount of time needed to execute that function. `Perf` makes use of statistical sampling to collect profile data,

Table 1: Comparison of *classical* FFD and profiled-based algorithm results for 13 images based on the Dice Similarity Coefficient (DSC) value.

Image #	Dimension	DSC
1	256x256x35	0.97262
2	256x256x120	0.95407
3	256x256x70	0.96194
4	256x256x70	0.96071
5	256x256x70	0.97167
6	256x256x120	0.93503
7	256x256x70	0.94767
8	256x256x70	0.95950
9	256x256x70	0.96650
10	256x256x120	0.97314
11	256x256x70	0.96029
12	256x256x70	0.95365
13	256x256x120	0.96493

thereby generating an interruption at regular time intervals. All processes running on the CPU are identified by `perf`, which then captures all relevant information such as the program counter, and CPU core number; next, it writes all of this data to an output file called `perf.data`. Additionally, `gprof` and `perf` runtime routines gather accurate call counts that combined with a post-processing version of the algorithm under analysis lead to a table where the number of calls to each function is presented, as well as the percentage, the amount of time spent in such function, and the average time per call.

4 Results and discussion

This section provides results of experiments aimed at getting useful profiling information, accumulating samples producing statistically meaningful results of the FFD algorithm under study using images of the MS Longitudinal Challenge dataset.

4.1 Algorithm Evaluation

The implementation was profiled using 13 images, and then the accuracy was evaluated by comparing the registration results with those obtained using a *classical* FFD implementation [14] by performing quantitative analysis using Dice Similarity Coefficient. The comparative results are presented in Table 1.

[14] An executable version of the used FFD algorithm for comparison purpose can be downloaded from Daniel Rueckert's webpage: `http://www.doc.ic.ac.uk/~dr`.

Table 2: Means and standard deviations of the runtime (in seconds) required by the profiled-based FFD's algorithm implementation.

Image #	Dimension	Runtime
1	256x256x35	73.08411 ± 0.05945
2	256x256x120	79.00041 ± 0.07101
3	256x256x70	74.08051 ± 0.01961
4	256x256x70	73.48618 ± 0.01920
5	256x256x70	74.20270 ± 0.01393
6	256x256x120	79.00217 ± 0.07294
7	256x256x70	74.68039 ± 0.01279
8	256x256x70	74.99707 ± 0.01484
9	256x256x70	73.94009 ± 0.01752
10	256x256x120	79.07080 ± 0.07590
11	256x256x70	74.08260 ± 0.01220
12	256x256x70	74.83009 ± 0.01522
13	256x256x120	79.00239 ± 0.08677

4.2 Computation time evaluation

To evaluate the benefit of a profile-based implementation regarding computer performance, required runtime was investigated. Each experiment was executed fifty times on each image; then the mean and standard deviation values of the time required to process the profiled-based algorithm were computed. All input images were performed, and it was included the time spent to load the data into the main memory system until the end of the registration process, i.e. until when the resultant image was produced. These results are presented in Table 2.

4.3 Performance analysis

As to performance analysis, it should be noted that the profiling tool collects data while monitoring performance counters, hardware interruptions, and operating system calls. Profiling tools periodically interrupt the kernel of the operating system to record a new sample and then the samples are stored in a ring buffer, generating overhead. `perf` mitigates sampling overhead thereby enforcing sampling buffer locality when `perf` creates one instance of the event on each CPU; then, the events are effectively measured when that CPU executes each thread. All the samples are aggregated into a single output file once all profiles are run. In the experiments conducted in here, the sampling mode in `perf` was used to trace the FFD algorithm events in real-time; `perf` generated output files of dozens of megabytes (for experiments with 2, 4, and 8 threads), as indicate in Table 3. This considerable big data size is because `perf` depends on the adopted frequency - here, a rate of 4000 samples per second was used - in which events are recorded, resulting in higher overhead and larger output files. However, *gprof* generates output files with

hundreds of kilobytes - 768 KB, mainly because the output file contains a histogram of program counter samples and the arc table.

Table 3: File sizes, indicated in megabytes (MB), generated by `perf` according to images dimension and the developed OpenMP-based implementation using different number of threads.

Image #	**Dimension**	**Number of threads**			
		1 Thread	2 Threads	4 Threads	8 Threads
1	256x256x35	9.65	10.23	12.24	25.41
2	256x256x120	12.40	13.16	18.91	28.74
3	256x256x70	11.08	11.83	13.02	27.53
4	256x256x70	17.74	18.48	22.31	25.42
5	256x256x70	36.32	32.37	61.61	25.45
6	256x256x120	8.61	8.92	9.84	25.63
7	256x256x70	9.16	9.45	25.53	33.02
8	256x256x70	7.85	8.23	11.33	30.56
9	256x256x70	12.04	12.51	15.92	30.22
10	256x256x120	24.07	25.04	31.32	50.58
11	256x256x70	18.44	19.14	25.81	52.37
12	256x256x70	12.91	13.60	14.64	31.66
13	256x256x120	20.51	21.20	33.58	46.05

In order to extract performance statistics and also record the arc in the call graph, the collected data were analyzed. This graph represents information intuitively employing a visual map from a collection of hierarchical data in order to quickly facilitate the understanding of large amounts of collected data [40–42]. Call graph represents time-consuming functions and the number of times the functions were invoked. By analyzing the call graph sample of the image registration algorithm under study, the graph shown in Fig. 2 was generated, which includes the time propagated for each function from its descendants, and the number of times each function was called.

The built call graph displays the descendants as well as the caller of each function, including the time propagated to each routine from its descendants. The significant entries of the call graph profile are the entries depicted by means of gray numbered circles in Fig. 2: the name of the caller function is represented by element 1; the percentage of the runtime accounted by the algorithm's function and its descendants is indicated by element 2; element 3 concerns the time regarding different meanings depending on whether it is the primary function for that section, the function's caller or descendant functions. In the first case, time shows the time spent on the function during the execution of the algorithm. In the second case, it indicates the amount of the first self-time function being propagated to that caller, based on the percentage of calls to the primary function made by that caller. Finally, for descendant functions, it represents the amount of that descendant function's self-time being propagated to the primary function based on the percentage of calls made to that function by the primary function; element 4 regards the number of times that

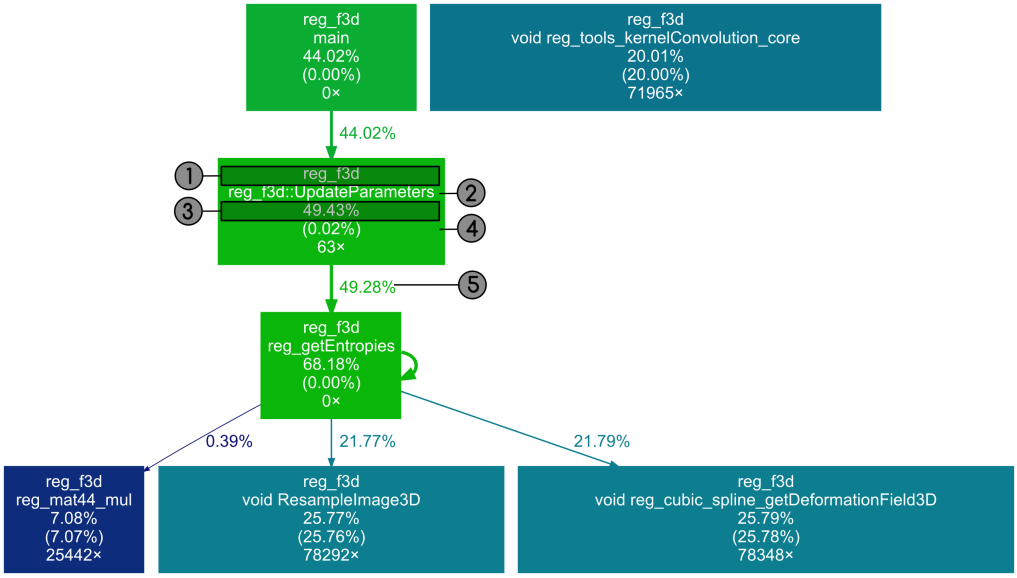


Figure 2: Call graph generated by `perf` representing the most often called functions in the studied image registration algorithm.

function was called; and finally, element 5 is related to the accumulated percentage of time running a function and propagated for each descendant function. All the information in Fig. 2 refers to the primary function of the studied nonrigid image registration algorithm.

The built call graph is helpful in evaluating the algorithm's performance and identifying its bottlenecks. Taking full advantage of profiling tools requires to focus on the analysis of the relevant parts of the algorithm execution, making the experiments easier to understand. Profiling tools identified that the function `reg_getEntropies` is responsible for 68% of the total running time (56.90 seconds), meaning that joint histogram filling is the main time-consuming task within this function. The other costly functions identified by the profiling tools were:

- `reg_cubic_spline_getDeformationField3D`, which generates the deformation field: a lattice of equally spaced control points is defined over the reference image using cubic B-splines;
- `ResampleImage3D`, which computes the value $I_s(T(x))$ for every pixel x, or voxel in 3D, inside the reference image. In this case, the computational complexity is linearly dependent on the number of pixels/voxels in the reference image;
- `UpdateParameters`, which assesses the quality of a registration using a cost function such as mutual information. In order to achieve the perfect registration between two images, transformation parameters are optimized iteratively.

The runtime was obtained by running the algorithm implementation fifty times and calculating the average of the time elapsed, as reported by each profiling tool. In all cases, the execution times for different runs of the implementation were remarkably consistent. The time-consuming functions iterate a hundred times and making them desirable parallelization targets. Based on the massive amount of work it performs, the studied algorithm exhibits a high degree of parallelism, since the algorithm iterates until convergence, aiming to ensure the best possible registration.

For the performance analysis of our parallel implementation, a benchmark problem was defined, which was qualified to evaluate the performance in sequential as well as in parallel execution. Then, we studied the effect of using a different number of physical cores on the performance of the multi-threaded developed algorithm. For a fixed number of cores, we used an equal number of threads for execution; that is, one thread for each core. The costly function `reg_getEntropies`, was implemented using OpenMP.

All the experiments previously performed were repeated and then compared using different degrees of parallelism: 1, 2, 4 and 8 threads. As shown in Fig. 3, the experiments using the developed parallel OpenMP-based implementation revealed a considerably reduction in the runtime of the non rigid image registration algorithm relatively to the single-thread implementation. This confirmed that profiling tools could help programmers quickly identify critical bottlenecks.

Fig. 4 depicts the performance gain of the parallel OpenMP-based implementation of the nonrigid image registration algorithm under study scales almost exponentially, and that the runtime of the parallel implementation achieved about seven times faster than the single thread-based implementation. The parallel-based implementation used one thread for each core for the execution.

5 Conclusions and future research

The need for parallelization is continuously increasing as almost all computing devices have multi-core processors, the applications are becoming more and more complex and demanding and the involved data is getting bigger and bigger. However, writing parallel code is still one of the biggest challenges for many programmers, because of the learning curve required for coding applications in parallel design, reaching a complete understanding of advanced concepts relating to memory hierarchy and the optimal (and shortest) data paths in computer systems. Among many efforts to reduce the burden of parallel programming, we mainly focused on support from profiling tools. As our findings suggest, profiling tools can be highly effective detecting and evaluating performance

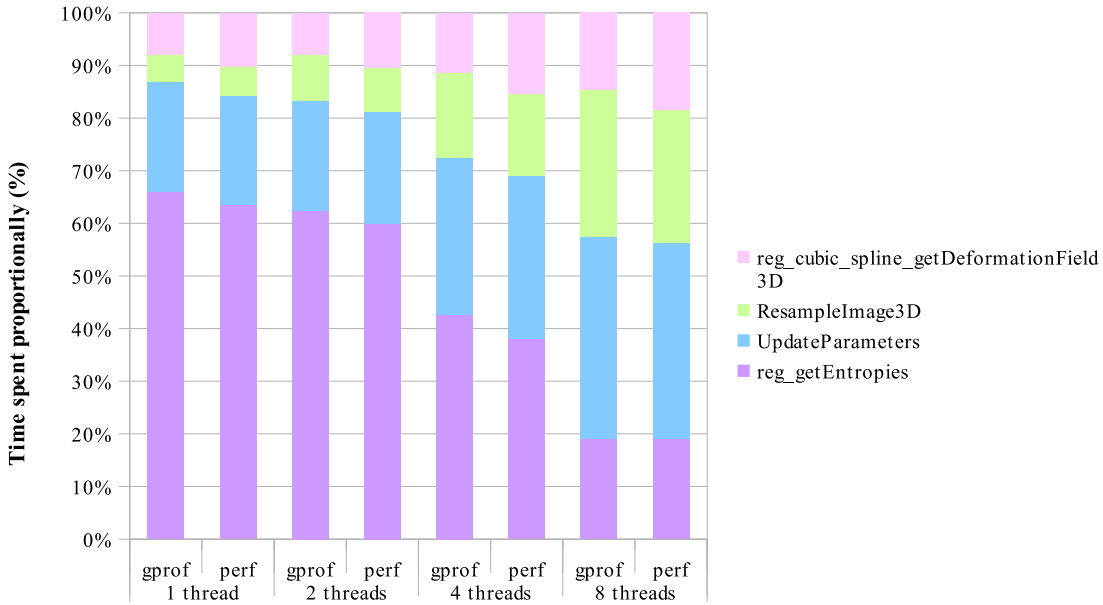


Figure 3: Proportionality of the time-consumption functions detected by the profiling tools `perf` and `gprof` using the developed OpenMP-based implementation of the FFD algorithm.

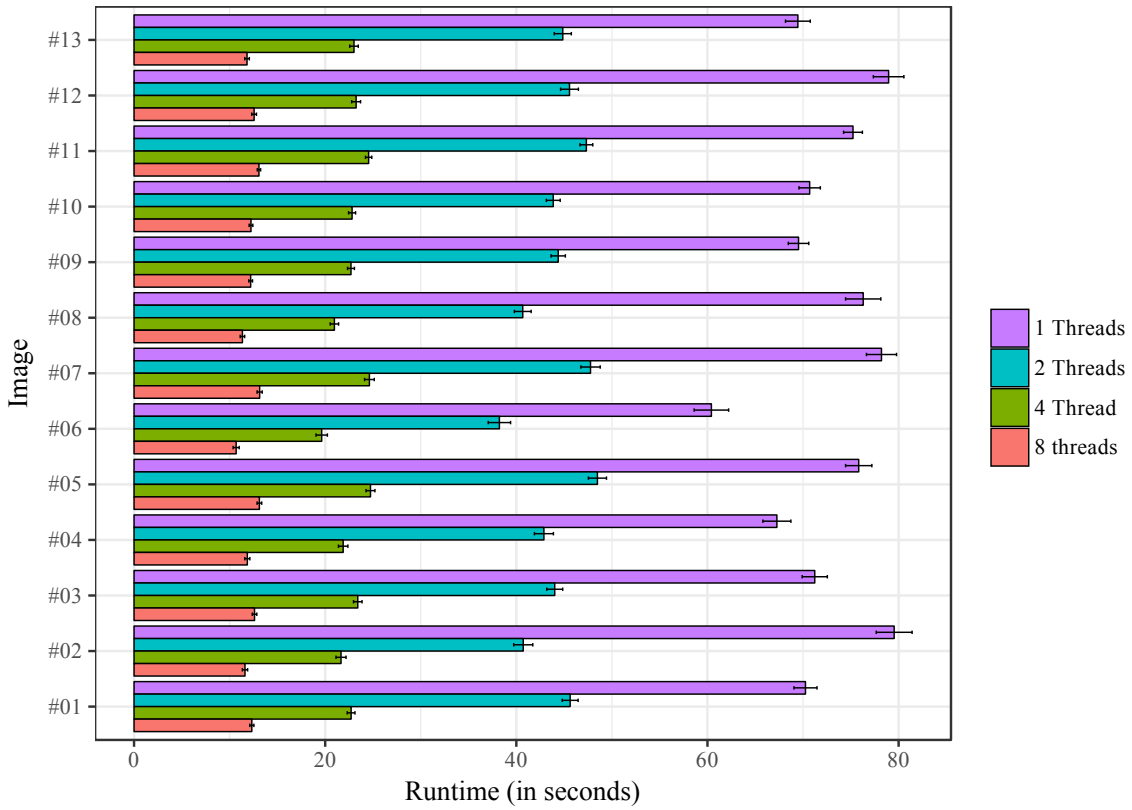


Figure 4: Means and standard deviations of runtime spent for running the developed OpenMP-based implementation of the FFD algorithm under study.

bottleneck snippets in a nonrigid image registration algorithm based on FFD, providing a low-impact method for gathering useful information.

The developed parallel OpenMP-based implementation was compared against the corresponding single thread-based implementation in several experiments. The parallelization of the costly functions of the FFD algorithm reduced the runtime by up to 7 times compared to the single thread-based implementation.

In conclusion, the proposed parallelization based on profiling tools substantially improved the runtime performance of the studied nonrigid image registration algorithm. This will facilitate medical practitioners and researchers, who commonly rely on image registration to label anatomical data, identify diseases, compare patient images or image sequences and perform patient follow-up, which therefore, makes substantially accelerated nonrigid image registration solutions accessible to a broader audience.

In future work, we will further develop the time-consuming functions already detected in this study, which can be made more efficient, and further speedups should be possible using more sophisticated data-parallel algorithms. We plan to optimize them by using heterogeneous parallel computing platforms based on GPUs. Additional challenges need to be addressed; for instance, the issue in shared memory systems of protecting simultaneous data access in order to avoid data inconsistency and errors, load balancing, and the efficient management of reading/writing data to massive data units. These challenging elements are all critical for achieving efficiency and the maximum performance possible in the underlying architecture.

6 Acknowledgments

The first author gratefully thanks for the support given the following: the Universidade do Estado de Mato Grosso (UNEMAT) of Brazil, and the National Council for Scientific and Technological Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq), process 234306/2014-9 grant under reference #2010/15691-0.

References

[1] S. P. P. Parraguez. *Fast and Robust Methods for Non-rigid registration of medical images*. PhD thesis, 2015.

[2] T. Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum. 3D nonrigid registration via optimal mass transport on the GPU. *Medical Image Analysis*, 13(6):

931–940, 2009. ISSN 1361-8415. doi: http://dx.doi.org/10.1016/j.media.2008.10.008.

[3] T. Rohlfing and J. Maurer, C.R. Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees. *IEEE Transactions on Information Technology in Biomedicine*, 7(1):16–25, 2003. ISSN 1089-7771. doi: 10.1109/TITB.2003.808506.

[4] F. P. Oliveira and J. M. R. Tavares. Medical image registration: a review. *Computer Methods in Biomechanics and Biomedical Engineering*, 17(2):73–93, 2014. doi: https://doi.org/10.1080/10255842.2012.670855.

[5] P. Snape, S. Pszczolkowski, S. Zafeiriou, G. Tzimiropoulos, C. Ledig, and D. Rueckert. A robust similarity measure for volumetric image registration with outliers. *Image Vision Comput.*, 52(C):97–113, August 2016. ISSN 0262-8856. doi: 10.1016/j.imavis.2016.05.006. URL `https://doi.org/10.1016/j.imavis.2016.05.006`.

[6] F. E.-Z. A. El-Gamal, M. Elmogy, and A. Atwan. Current trends in medical image registration and fusion. *Egyptian Informatics Journal*, 17(1):99 – 124, 2016. ISSN 1110-8665. doi: https://doi.org/10.1016/j.eij.2015.09.002. URL `http://www.sciencedirect.com/science/article/pii/S111086651500047X`.

[7] O. Dandekar and R. Shekhar. FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions. *IEEE Transactions on Biomedical Circuits and Systems*, 1(2):116–127, 2007. ISSN 1932-4545. doi: 10.1109/TBCAS.2007.909023.

[8] S. K. Warfield, F. A. Jolesz, and R. Kikinis. A high performance computing approach to the registration of medical imaging data. *Parallel Computing*, 24: 1345–1368, 1998. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(98)00061-1.

[9] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91 – 108, 1996. ISSN 1361-8415. doi: https://doi.org/10.1016/S1361-8415(96)80007-7. URL `http://www.sciencedirect.com/science/article/pii/S1361841596800077`.

[10] M. Salomon, F. Heitz, G.-R. Perrin, and J.-P. Armspach. A massively parallel approach to deformable matching of 3D medical images via stochastic differential

equations. *Parallel Computing*, 31(1):45–71, 2005. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2004.12.003.

[11] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, and S. Ourselin. Fast free-form deformation using graphics processing units. *Computer Methods and Programs in Biomedicine*, 98(3):278 – 284, 2010. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2009.09.002. URL `http://www.sciencedirect.com/science/article/pii/S0169260709002533`. HP-MICCAI 2008.

[12] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Nonrigid registration using free-form deformations: application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8):712–721, Aug 1999. ISSN 0278-0062. doi: 10.1109/42.796284.

[13] R. Palomar, J. Gómez-Luna, F. A. Cheikh, J. Olivares-Bueno, and O. J. Elle. High-performance computation of bézier surfaces on parallel and heterogeneous platforms. *International Journal of Parallel Programming*, May 2017. ISSN 1573-7640. doi: 10.1007/s10766-017-0506-1. URL `https://doi.org/10.1007/s10766-017-0506-1`.

[14] J. Shackleford, N. Kandasamy, and G. Sharp. *High Performance Deformable Image Registration Algorithms for Manycore Processors*. Morgan Kaufmann Publishers Inc., 2013. ISBN 0124077412, 9780124077416. doi: https://doi.org/10.1016/B978-0-12-407741-6.00007-4.

[15] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. A survey of medical image registration on multicore and the GPU. *IEEE Signal Processing Magazine*, 27(2):50–60, 2010. ISSN 1053-5888. doi: 10.1109/MSP.2009.935387.

[16] N. D. Ellingwood, Y. Yin, M. Smith, and C.-L. Lin. Efficient methods for implementation of multi-level nonrigid mass-preserving image registration on GPUs and multi-threaded CPUs. *Computer Methods and Programs in Biomedicine*, 127:290 – 300, 2016. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2015.12.018. URL `http://www.sciencedirect.com/science/article/pii/S016926071530033X`.

[17] Z. Li, R. Atre, Z. Huda, A. Jannesari, and F. Wolf. Unveiling parallelization opportunities in sequential programs. *Journal of Systems and Software*, 117:282 – 295, 2016. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2016.

03.045. URL http://www.sciencedirect.com/science/article/pii/S016412121630005X.

[18] S. Rul, H. Vandierendonck, and K. D. Bosschere. A profile-based tool for finding pipeline parallelism in sequential programs. *Parallel Computing*, 36(9): 531 – 551, 2010. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2010.05.006. URL http://www.sciencedirect.com/science/article/pii/S0167819110000840.

[19] S. L. Graham, P. B. Kessler, and M. K. McKusick. gprof: A call graph execution profiler. *ACM SIGPLAN Notes*, 39(4):49–57, April 2004. ISSN 0362-1340. doi: 10.1145/989393.989401. URL http://doi.acm.org/10.1145/989393.989401.

[20] M. Dimakopoulou, S. Eranian, N. Koziris, and N. Bambos. Reliable and efficient performance monitoring in Linux. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13. IEEE Press, 2016. ISBN 978-1-4673-8815-3. URL http://dl.acm.org/citation.cfm?id=3014904.3014950.

[21] E. Rohou. Tiptop: Hardware performance counters for the masses. In *2012 41st International Conference on Parallel Processing Workshops*, pages 404–413, Sept 2012. doi: 10.1109/ICPPW.2012.58.

[22] T. Ball and J. R. Larus. Optimally profiling and tracing programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1319–1360, July 1994. ISSN 0164-0925. doi: 10.1145/183432.183527. URL http://doi.acm.org/10.1145/183432.183527.

[23] M. Schulz and B. R. de Supinski. *Practical Differential Profiling*, pages 97–106. Springer, 2007. ISBN 978-3-540-74466-5. doi: 10.1007/978-3-540-74466-5_12. URL http://dx.doi.org/10.1007/978-3-540-74466-5_12.

[24] J. M. Spivey. Fast, accurate call graph profiling. *Software: Practice and Experience*, 34(3):249–264, March 2004. ISSN 0038-0644. doi: 10.1002/spe.562. URL http://dx.doi.org/10.1002/spe.562.

[25] A. Li, A. Kumar, Y. Ha, and H. Corporaal. Correlation ratio based volume image registration on GPUs. *Microprocessors and Microsystems*, 39(8):998 – 1011, 2015. ISSN 0141-9331. doi: https://doi.org/10.1016/j.micpro.2015.04.

002. URL http://www.sciencedirect.com/science/article/pii/S0141933115000459.

[26] L. Shi, W. Liu, H. Zhang, Y. Xie, and D. Wang. A survey of GPU-based medical image computing techniques. *Quantitative Imaging in Medicine and Surgery*, 2(3), 2012. URL http://qims.amegroups.com/article/view/1079.

[27] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. *Computer Methods and Programs in Biomedicine*, 99(2):133 – 146, 2010. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2009.11.004. URL http://www.sciencedirect.com/science/article/pii/S0169260709002946.

[28] S. Mittal and J. S. Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys*, 47(4):69:1–69:35, July 2015. ISSN 0360-0300. doi: 10.1145/2788396. URL http://doi.acm.org/10.1145/2788396.

[29] F. Gebali. *Algorithms and Parallel Computing*. John Wiley & Sons, 2011. ISBN 978-0-470-90210-3.

[30] A. Vadja. *Programming Many-Core Chips*. Springer, 2011. ISBN 978-1-4419-9738-8. doi: 10.1007/978-1-4419-9739-5.

[31] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte. Medical image processing on the GPU - past, present and future. *Medical Image Analysis*, 17(8):1073–1094, 2013. ISSN 1361-8415. doi: 10.1016/j.media.2013.05.008.

[32] L. Gong and C. Kulikowski. High-performance medical imaging informatics. *Methods of Information in Medicine*, 51(3):258 – 9, 2012. ISSN 0026-1270.

[33] L. Meng. Acceleration method of 3D medical images registration based on compute unified device architecture. *Bio-medical materials and engineering*, 24 (1):1109–1116, 2014. doi: 10.3233/BME-130910.

[34] G. E. Christensen. MIMD vs. SIMD parallel processing: A case study in 3D medical image registration. *Parallel Computing*, 24:1369–1383, 1998. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(98)00062-3.

[35] J. Rohrer and L. Gong. Accelerating 3D nonrigid registration using the cell broadband engine processor. *IBM Journal of Research and Development*, 53(5), 2009. ISSN 0018-8646. doi: 10.1147/JRD.2009.5429078.

[36] R. J. Lapeer, S. K. Shah, and R. S. Rowland. An optimised radial basis function algorithm for fast non-rigid registration of medical images. *Computers in Biology and Medicine*, 40(1):1–7, JAN 2010. ISSN 0010-4825. doi: 10.1016/j.compbiomed.2009.10.002.

[37] R. Mafi and S. Sirouspour. GPU-based acceleration of computations in nonlinear finite element deformation analysis. *International Journal for Numerical Methods in Biomedical Engineering*, 30(3):365–381, 2014. ISSN 2040-7939. doi: 10.1002/cnm.2607.

[38] A. Carass, S. Roy, A. Jog, J. L. Cuzzocreo, E. Magrath, A. Gherman, J. Button, J. Nguyen, F. Prados, C. H. Sudre, M. J. Cardoso, N. Cawley, O. Ciccarelli, C. A. Wheeler-Kingshott, S. Ourselin, L. Catanese, H. Deshpande, P. Maurel, O. Commowick, C. Barillot, X. Tomas-Fernandez, S. K. Warfield, S. Vaidya, A. Chunduru, R. Muthuganapathy, G. Krishnamurthi, A. Jesson, T. Arbel, O. Maier, H. Handels, L. O. Iheme, D. Unay, S. Jain, D. M. Sima, D. Smeets, M. Ghafoorian, B. Platel, A. Birenbaum, H. Greenspan, P.-L. Bazin, P. A. Calabresi, C. M. Crainiceanu, L. M. Ellingsen, D. S. Reich, J. L. Prince, and D. L. Pham. Longitudinal multiple sclerosis lesion segmentation: Resource and challenge. *NeuroImage*, 148:77 – 102, 2017. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2016.12.064. URL `http://www.sciencedirect.com/science/article/pii/S1053811916307819`.

[39] D. Kirk and W.-M. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, 2010. ISBN 978-0-12-381472-2.

[40] C. P. Bezemer, J. Pouwelse, and B. Gregg. Understanding software performance regressions using differential flame graphs. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 535–539, March 2015. doi: 10.1109/SANER.2015.7081872.

[41] B. Gregg. The flame graph. *Queue*, 14(2):91–110, mar 2016. ISSN 1542-7730. doi: 10.1145/2927299.2927301. URL `http://doi.acm.org/10.1145/2927299.2927301`.

[42] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983. ISSN 00031305. URL http://www.jstor.org/stable/2685881.