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

Carlos A. S. J. Gulo, CNPq National Scientific and Technological Development Council, Research Group PIXEL - UNEMAT, Brazil, Programa Doutoral em Engenharia Informática, Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial, Faculdade de Engenharia, Universidade do Porto, Portugal, sander@unemat.br

Antonio C. Sementille, Departamento de Ciências da Computação, Faculdade de Ciências, Universidade Estadual Paulista-UNESP, Brazil, antonio.sementille@unesp.br

João Manuel R. S. Tavares, Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial, Departamento de Engenharia Mecânica, Faculdade de Engenharia, Universidade do Porto, Portugal, tavares@fe.up.pt

Abstract Common image segmentation methods, to a large extent, are computationally expensive, particularly when run on large medical datasets, and require powerful hardware to achieve image-based diagnosis in real-time. For a medical image segmentation algorithm that is based on an active contour model, our work presents an efficient approach that detects computationally-intensive functions and adapts the implementation for improved performance. We employ profiling methods that assess algorithm performance taking into account the overall cost of execution, including time, memory access, and performance bottlenecks. We apply performance analysis techniques commonly available in traditional computing operating systems, which obviates the need for new setup or measurement techniques ensuring a short learning curve. Thus, the approach is well-suited for the community of researchers in the area of medical image processing and analysis. The article presents guidelines to aid researchers in *a)* using profiling tools and *b)* detecting and checking potential optimization snippets in medical image segmentation algorithms by measuring overall performance bottlenecks. For the profiling tools used in this work, `gprof` and `perf`, both offered a similar cost in terms of execution time and gathered accurate data about the relationship between execution time and paths in the call graph. Call graph visualization provides developers a quick graphical overview of the execution time of codes as well as throughput in memory accesses, and performance bottlenecks, significantly easing the performance analysis.

Keywords Medical Image Processing and Analysis, Profiling Tools, Performance Analysis, High-performance Computing

1 Introduction

Image segmentation is an imperative operation in medical image analysis, being responsible for identifying and delineating key regions within an image. To achieve the best results, an adequate image segmentation is essential for several tasks, such as 3D vision, image registration, image classification, and interpretation [1].

Active contour models (ACM), or "snakes" as commonly known in the scientific community [2, 3, 4, 5, 6], offer an attractive approach to addressing contour detection problems. 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 an 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 [7]. However, this method entails complex calculations, and there is a need to develop new optimization strategies that reduce the required runtime. To this end, we can employ profiling methods which deployment has contributed effectively to identify and evaluate portions of code bound to consume excessive computational resources [5, 8].

Under normal operation, profiling tools can count the activation instances of a function accurately during runtime of the algorithm, indicating thus timing information of the function [9]. Profiling, at this level, is a useful technique to assist algorithm's optimization based on collecting and measuring data related to memory space requirement, frequency, and duration of the function calls, as well as time complexity of the algorithm. Many profiling tools such as `gprof` [9], `perf` [10], `tiptop` [11] and others [12, 13, 14] have been proposed to assist programmers in identifying performance bottlenecks when executing an algorithm on CPU under a given workload [10, 15].

In this work, we aim to identify time-consuming functions in the Chan-Vese ACM algorithm using profiling tools. The performance analysis based on profiling can be adapted to effectively reduce the processing time of the algorithm, making it suitable for real-time diagnosis. For this, we exploit available computational power commonly available on a typical computer. In this scenario, the term "performance" refers to the efficiency of computing systems when executing algorithms, including the factors of throughput, latency, and availability.

The article is mainly focused on medical image processing and analysis applications and provides important guidelines that support researchers in identifying time-consuming functions in their algorithms using profiling tools. The experimental findings show that profiling information can indicate majority bottlenecks in the algorithm. This study also provides insight into the significance of profiling data.

This is the first time that the chosen profiling tools are used to support the parallelization of a medical image segmentation algorithm. The findings contribute to a better understanding of image processing and analysis. Today, there is a steady rise in the number of images with high resolution that need to be processed and analyzed fast, e.g., in real-world clinical scenarios. Also, computers with multi-core processors are commonly available in medical environments with adequate computational power for efficient image processing and analysis. The insights achieved with the profiling process can aid in efficient implementation of these algorithms with substantial impact in the area. One major limitation is the sequential code is not parallelized automatically by our approach.

This article is organized as follows: Section 2 introduces the background concepts and methods that have been proposed to accelerate the processing of image segmentation algorithms. Section 3 presents methodologies commonly used in this research area; Section 4 details our findings and presents the discussion concerning the use of the profile information gathered about the medical image segmentation algorithm under study. Finally, Section 5 contains the conclusions and suggestions for future studies.

2 Background and Related Work

This section introduces two topics: the segmentation of medical images with a focus on the Chan-Vese ACM algorithm, and the profiling method used to pinpoint bottlenecks that present excessive CPU consumption. In addition, a literature review covering studies of image processing and analysis methods accelerated by high-performance computing architectures is presented.

2.1 Medical Image Segmentation

Segmentation is the technique of partitioning an image into its constituent homogeneous regions. Partitioning is commonly carried out based on the desired image feature(s), such as color, intensity, or texture [16, 4]. Medical image segmentation is crucial, for example, for the successful extraction of image features and their subsequent classification, and for aiding in the visualization by performing the detection process efficiently. Briefly, medical image segmentation has application in image-based diagnosis and monitoring, and planning and navigation during surgery [16, 3].

Image segmentation methods have been applied for partitioning images that are important for a medical point of view, and are acquired from a range of objects , e.g., lungs [17], skin lesions [18, 19], and vessels [20, 4]. Different imaging modalities have been used to acquire clinically useful information about anatomical structures including magnetic resonance, computed tomography, ultrasound, and others, and many image segmentation methods have been developed to segment the acquired images [4, 16, 17, 20, 21, 22].

Image segmentation techniques can be grouped into four categories: pixel-based, region-based, edge-based, and model-based [16]. Region and pixel-based techniques are based on the concept of the discontinuity of pixel values, whereas pixel valueśsimilarity is the basis of the edge-based techniques. Region-based techniques rely on similar patterns in intensity values within a region of neighboring pixels, including 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. For these methods, the focus is on finding discontinuities, i.e., points, lines and edges, concerning features such as color, intensity, or texture. In model-based techniques, image segmentation is framed as a statistical optimization problem [23].

By convention, image segmentation can be defined as the problem of finding partition(s) S_k of a dataset such that each partition $S_k \subset I$ represents a homogeneous region. The union of all such partitions 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 work in Kass et al. [2] introduced the concept of active contours which was widely accepted as a new approach to image segmentation. Of the main limitations of active contour models is the requirement for the initial contour being close to the boundary of interest. Moreover, active contours have difficulties to progressing into boundary concavities. The work in Chan and Vese [7] proposed an algorithm to address the limitations of the traditional snake model. The Chan-Vese algorithm derives from a compilation of two other techniques: the level set method, used in edge detection through a topological change of the curves, and 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 particular, the Chan-Vese algorithm achieved notable results in the segmentation of different objects with various shapes and with inner contours.

In many instances, the algorithm successfully detected and preserved the location of boundaries, even for objects which boundaries are not very smooth or defined by gradient. 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]. Dynamic equations govern the movement of an active contour model which performs successive minimization iterations of a given energy associated with the curve. The Chan-Vese ACM is based on variational methods; 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:

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

The solution to Equation 2 is found by successive iterations, and in 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 [23, 2]. The Chan-Vese method has been validated by a range of numerical results as reported in [7]. Algorithm **??** lists the pseudocode of the Chan-Vese algorithm.

2.2 Profiling Method

In this section, we review the use of profiling as means to measure the computation time of each function in an algorithm. Profiling is commonly used to discern the behavior of an algorithm and measure its performance by collecting relevant 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 [14, 24]. Profiling methods include instrumented, event-based, statistical, and simulation [9, 10, 8].

Performance analysis based on profiling consists of the following steps applied in order: instrumentation or modification of the algorithm to produce performance data, measuring important execution aspects that generates these data, data analysis, and finally visualization of the performance data [13] (Fig. 1).

2.2.1 Instrumentation

Instrumentation is the process of incorporating measurement code within an algorithm at compile time, aiming to have a precise measurement of execution times. This procedure adds a detailed listing of the running statistics to the object file and links the executable with standard libraries with profiling information enabled. However, it requires the availability of the source code and the compiler [13, 5].

When using a profiling method, a key consideration is to effect a minimum change in the algorithm's intended behavior. Monitoring algorithm runtime behavior involves instrumenting the binaries to record desired events; the system event data, in essence, keeps track of the interaction between programs and the hardware.

2.2.2 Measuring

Gathering profile data is the second step of the profiling method. In this step, we monitor hardware interrupts, operating system calls and performance counters [9, 25]. After performance data from one or more executions have been recorded, information linked with functions is extracted and stored in output files. In a properly implemented profiling process, gathering of profiling data does not interfere with the algorithm execution.

Performance counters are available in most modern processors, enabling the count of various hardware performance events such as clocks per cycle, 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 binaries which contain the execution profile are available in the output files called `perf.data` and `gmon.out`, for the `perf` and `gprof` profiler respectively. Data is analyzed to extract performance statistics, for instance, the number of times a function is called and the time

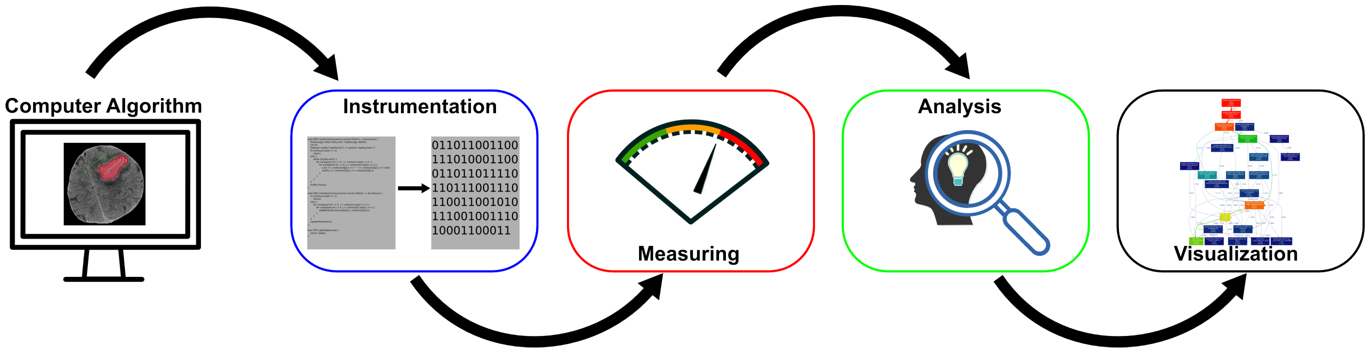

Figure 1. Diagram of the profiling method. Each part of the diagram shown is described in the text.

spent in each function. The profilers also record the arc in the call graph that activates the function [10, 9, 13]. The returning address of a function is used for identifying the source of the arc referred as *caller*, and the destination of the arc referred as *callee* [12].

For finding the most costly function in an algorithm, it is critical to collect the arcs of the dynamic call graph traversed during the algorithm execution. Hence, in the post-processing of the data, it is possible to visualize the call graph and to show the measures collected from the algorithm execution.

2.2.4 Visualization

The last step concerns visualization of the collected data. Call stack walking is a technique that shows the functioning internal to any algorithm without source code access. The technique can show functions being called and the CPU usage time for these function. The profilers, `gprof` and `perf` provide dynamic call graph information for all instrumented code snippets. A call graph is binary and sometimes is treated as a multi-graph, i.e., instead of simple relations, there can be multiple relations between the same two nodes. For example, it has relation over functions, or procedures defined in an algorithm [10, 9]. The edges represent the calls between the functions being run in the algorithm and their 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 [26]. On the other hand, several studies have been proposed to address the segmentation of medical images using high-performance computing [27, 28, 26]; 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. [29]. Many authors deploy approaches of image segmentation based on thresholding [30, 31, 32], clustering [33] and deformable models [34], on a PC-cluster [35, 36, 33], using graphics processing unites (GPUs) [37, 38, 39, 26, 30, 40, 41] or multi-core processors [33, 41, 42, 43].

Daggett and Greenshields [35] and Yeh and Fu [36] 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. [33] 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 [30, 39] and Bias Field Correction Fuzzy C-Mean [26]. However, the expensive computation required by the algorithms demanded optimization strategies in order to reduce the run-time; hence, Lamas-Rodríguez et al. [39] 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 [30] 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. [39] nor Balla-Arabé and Gao [30] 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. [26], the GPU implementation achieved real time in segmenting volume images.

Zhuge et al. [37, 38] 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. [40] 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. [41] 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, 44], see Section 2.2.

3.1 Experimental Setup

Our test and development infrastructure included a desktop computer with the Linux Debian 8 operating system, GNU gcc/g++ compiler 4.9.2, the application programming interface OpenMP 3.1, Visual Studio Code 1.57.1, the profilers `gprof` 2.25 and `perf` 3.16.7-ckt20. Also, we used `gprof2dot` [1], and `dot` [2] 2.38, 16 GB of RAM (DDR3-1600

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

[2] `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`

MHz), and an Intel(R) 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 using the support of a feature referred *hyper-threading technology*. Thus, effectively, we can choose to run from a single thread to a maximum of 8 threads considering if we use each core partially or fully.

3.2 Dataset

The study used Multiple Sclerosis (MS) images selected from the MS Longitudinal Challenge Data Set repository [45] which access is open for research purposes. We picked thirteen images from the initial dataset to study the effectiveness of the chosen segmentation method. The images were chosen randomly and were subjected to the same preprocessing, with the data acquired using a 3.0 Tesla MR imaging scanner from Philips Medical System, Best, The Netherlands. The parameters are as follows: 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 [45]. Figure **??** shows an example of a segmentation obtained with the algorithm under analysis.

3.3 Segmentation Evaluation

Dice Similarity Coefficient (DSC) is a statistical metric commonly used to evaluate the performance of the reproducibility of ground truth segmentation and the spatial overlap accuracy of automated probabilistic fractional segmentation. The DSC value is a simple and useful summary measure of spatial overlap, and can be applied to studies of reproducibility and accuracy in image segmentation [46, 47]. The DSC value ranges from zero indicating no spatial overlap between two segmentation results to one indicating a complete overlap. Thus, DSC measures the spatial overlap between two segmentation 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; X is the segmentation area obtained by the algorithm, Y is the area of the ground truth segmentation, and $X \bigcap Y$ is the overlapping area of the two segmentation.

3.4 Performance Evaluation

For measuring the speedup of the Chan-Vese algorithm, we focused on the runtime taken by each function in the algorithm, using the profiling tools: `gprof` and `perf`. We selected `gprof` and `perf` tools as they combine three profiling methods: instrumented, event-based, and statistical. `gprof`, commonly considered as easy to use and portable, is limited in scope; it is designed to produce a detailed call graph identifying the functions that call other functions and the frequency of the calls. Moreover, `gprof` provides information about the number of calls made to each function, the percentage of the total time spent in the function, and computes the time needed to execute that function. `perf`, on the other hand, makes use of statistical sampling to collect profile data, thereby generating an interruption at regular time points. `perf` can identify all processes running on the CPU; it captures relevant information such as the program counter, the CPU core. It then writes these data to an output file named `perf.data`.

We compiled a single thread-based implementation of the Chan-Vese algorithm to create a working executable. The following parameters were applied at compile time: `-fno-omit-frame-pointer` that enables frame pointer analysis, `-g` that generates symbol information and enables source code analysis, and `-pg` that compiles and links the source code with profiling information enabled; the monitor function `mcount` is inserted before each function call.

The compiler parameter `-pg` causes each profiled function to call the monitoring function `mcount` as one of its first operations. `mcount` notes its own return address that falls in the profiled function which is the destination of an arc in the call graph. The monitoring function also identifies the source of the arc from the return address of the function. Arcs representing invocations in the same function are termed as cycles. When a child function is a member of a cycle, the time indicated is the fraction of the time for the whole cycle. Self-recursive routines have their calls failed into calls from the outside and self-recursive calls; thus, only the outside calls affect the propagation of time. Importantly, the algorithm calls `libc-2.19` are related to the `C` runtime library, and that is not uncommon to use significant amounts of time in a runtime library and not in the algorithm code itself.

4 Results and discussion

We performed certain experiments with the aim of collecting useful profiling information, accumulating data that produce statistically meaningful observations, and reducing measurement errors of the Chan-Vese algorithm. We report results from these experiments in this section. We segmented each test image with the Chan-Vese algorithm. The objective of the reported work, however, is not to assess the accuracy of the used segmentation algorithm. Rather, we focused on measuring the performance of functions on the Chan-Vese implementation and for its speed-up with the multi-thread implementation.

4.1 Algorithm Evaluation

We performed a quantitative evaluation to analyze the segmentation results, i.e., the ground truth of the segmented regions was used to confirm whether each lesion present in the thirteen images was correctly segmented. We compared the segmented images against the ground truths using Dice Similarity Coefficient. Results are shown in Table **??**.

4.2 Runtime Evaluation

For performance evaluation, we measured the running time using a C++ function for all the reported experiments. Each experiment was executed fifty times for each image. Mean and standard deviation values of the time required to segment each input image were calculated. The computations account for the time spent to load the data into the system memory until the end of the segmentation process, when the resultant image is produced. The results are shown in Table **??**.

4.3 Performance Analysis

Gathering profile data is the next step in the process. We collect data while monitoring hardware interrupts, operating system calls and performance counters. Profiling tools periodically record new samples by interrupting the operating system kernel and save these samples in a ring buffer, generating overhead. `perf` mitigates sampling overhead by enforcing sampling buffer locality; it creates one instance of an event on each CPU. 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. For the experiments in this study, we used the sampling mode in `perf` to trace Chan-Vese algorithm events in real time. The output files generated by `perf` are larger in size than the ones resulting with the `gprof`. With `gprof`, output file size is approximately 7.9 KB (for experiments with 2, 4, and 8 threads) and 16 KB (for experiments with a single thread), in contrast to the `perf` profile where the output file size is in the order of dozens of megabytes. Table **??** lists these results. The great difference in profiling data sizes is primarily because of the manner in which data is stored. The output file with `gprof` stores a histogram of algorithm counter samples and the arc table. The resulting file size with `perf`, on the other hand, depends on the sampling frequency at which events are recorded. With a typical rate of 4000 samples per second, the process generates big overhead and larger output files.

We analyzed the data to extract performance statistics. Mainly, we record the arc in the call graph that activates each profiled function. Call graph represents time-consuming functions and the number of times such functions are invoked. We analyzed the call graph sample for segmentation of image #1, and generated the call graph shown in Fig. 3. This call graph incorporates the time required for each function from its descendants as well as the number of times each function is called.

For a given function, the call graph displays its children as well as parents (the call sites that invoke this function). The call graph also includes the higher level functions that consume large parts of the total execution time in the functions that they call. In the context of this study, children refer to the functions that are called by another function (parent). 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 accounts for a time that depends on whether it is the primary function for that section, the functions caller or child functions. In the first case, time is the actual function execution time during running of the algorithm. In the second case, it indicates the amount of the self-time 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 functions self-time being propagated to the primary function based on the percentage of calls made to the descendant by the primary function. The next item, item 4 is represents the number of times that function was called; and finally, item 5 is related to the accumulated percentage of time running a function, taking into account propagation for each descendant function.

The call graph helps focus the analysis on the relevant parts of the algorithm execution, making the experiments easier to understand. From the call graph, it is easy to see that the `main` function called the `ChanVeseSegmentation` function, which then called the functions `GetCVC`, `ReinitPhi`, `Image::data`, `min`, and `max`.

Function `ReinitPhi` locally computes the signed distance function to its zero level set. Our method identified `ReinitPhi` as the most called function with the maximum required running time, approximately 80% of the total running time, i.e., 12.90 seconds (see Algorithm **??**, lines 6 to 12). Function `GetCVC` computes the coefficients needed in the Chan-Vese algorithm for the level set function. `Image:data` assigns a point to minimal energy neighborhood (Algorithm **??**, 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 (Algorithm **??**, line 9).

Fig. 4 shows a percentage chart of the most used functions in the studied segmentation algorithm. The measurement depicted reflects the time spent on each function based on counter events. All times are obtained by running Chan-Vese algorithm fifty times, saving the time elapsed in each run as reported by the profiling tool, and calculating the average of these times. In all cases, the execution times for different runs of the implementation were quite consistent. Functions `ReinitPhi` and `GetCVC` appeared the most frequent in the run stack; responsible for taking up the processor time for 23.50 seconds (in exclusive usage), which implies 98.60% of the total runtime. In particular, these functions accounted for a hundred iterations, making them interesting targets for parallelization. The algorithm is responsible for large computations, and since each element on

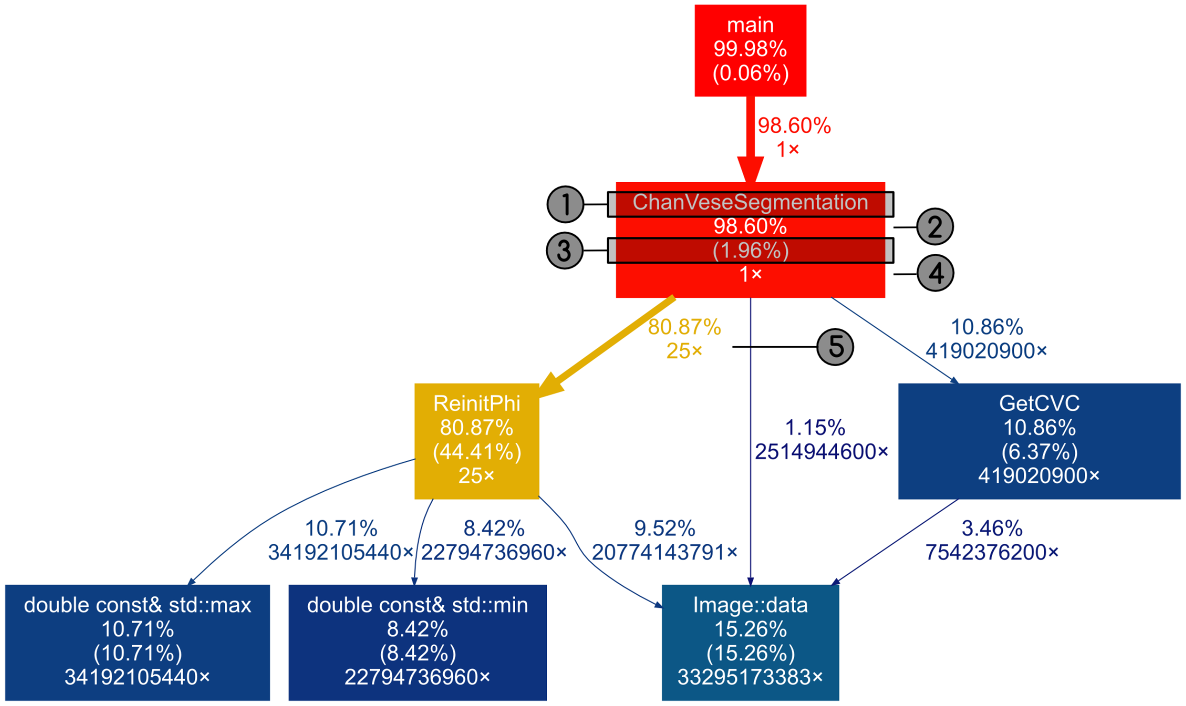

Figure 3. Call graph generated by `perf` representing the most often functions called by the Chan-Vese algorithm.

the input image can be computed independently, the algorithm can greatly benefit from a high degree of instruction parallelism.

In general, the segmentation algorithm is considered computationally-intensive due to the high number of iterations and computations per iteration. The suggestions generated by the profiling tools have pointed to the most promising targets for efficiency gains. The function that iterates over the lines of the input image and solves Equation 2 represents the highest computation, and takes up the most execution time.

4.4 Effect of the Number of Used Cores

We discuss how using different number of physical cores impact the performance of the multi-threaded Chan-Vese segmentation algorithm. With a given number of cores, we allocate equal number of execution threads to each core. We implemented `ReinitPhi`, the most intensive function, using OpenMP. The experiments performed previously in sequential execution mode were repeated under a parallel mode. We evaluated the performance in different degrees of parallelism, i.e., using 1, 2, 4 and 8 threads. As shown in Fig. **??**, using the parallel OpenMP-based implementation, there is significant reduction in algorithm runtime when compared to the single thread based implementation.

For large images with dimensions over 768×576 pixels, we see a noticeable speed-up of the OpenMP-based implementation. In particular, Fig. **??** suggests performance scale up is almost exponential, being the processing time of the parallel implementation approximately 7 times faster than a single thread based implementation. Our results confirm that multi-core processors have the capacity to lend high performance gains when using parallel OpenMP-based implementations. For a fixed number of cores, we used an equal number of threads for the execution, one thread on each core. Fig. **??** clearly shows that for each image size the execution time decreases as we increase the number of cores. In summary, the insights gained with the profiling tools output aids in better implementations, in particular, writing parts of the

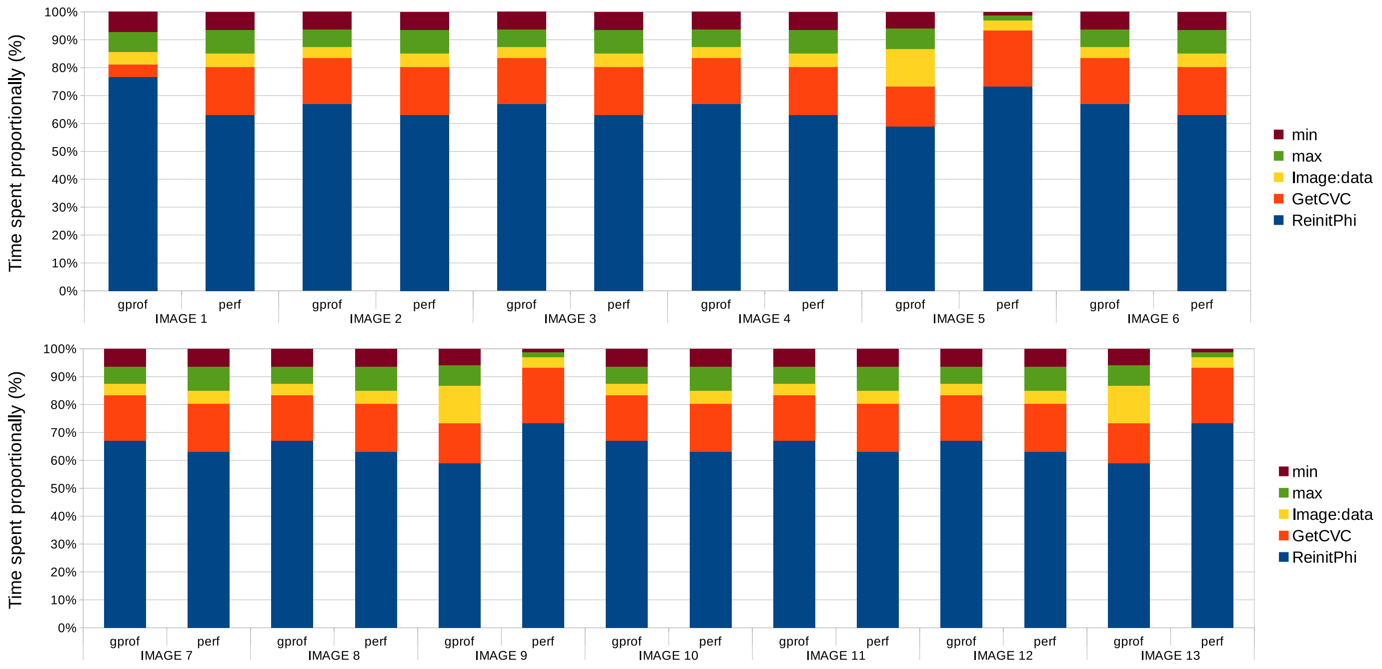

Figure 4. Most time-consuming functions detected by the profiling tools `perf` and `gprof`.

code with parallelized implementation. This computational parallelization can significantly increase the application performance.

5 Conclusion and Future Work

In this work, we described an approach that uses profiling tools to detect and evaluate code snippets that present as performance bottlenecks in a particular implementation. The implementation that we considered is an image segmentation algorithm based on the active contour, namely the Chan-Vese algorithm. We developed a parallel OpenMP-based implementation of the same algorithm using insights achieved with the profiling process. We compared the performance of the parallel implementation against a base-line single threaded implementation in several experiments. Parallelizing the implementation of the costly algorithm function reduced the runtime by up to 7 times when compared to a single thread based implementation.

The novelty of the proposed method lies in using profiling as means to efficient deployment of medical image processing and analysis applications. The method presented an effective solution for the applications with high processing demands. The output of the profiling method is a detailed profile that is combined with the source level information to identify and evaluate performance bottleneck snippets in the algorithm. The approach detected the available parallelism targets, as well as, substantially reduced the total time needed to parallelize the sequential code.

Our findings confirmed that parallel computing can provide substantial acceleration in processing speeds. The processing time decreased in all cases, as the number of threads increased. Many factors impact the improvement in the execution including compiler optimizations, runtime support, data layout, operating system noise, workload balancing etc. In addition, it may depend on the regions needing to be merged in the structure of the graph of the input image. Under parallel mode, any thread that completes execution must wait for the last running thread. This leads to slack time, i.e., keeping the operating system from possibility of executing another task in the meantime. Nevertheless, the time improvement with parallelization is significant and promising.

For future work, we aim to optimize the time-consuming functions detected by the presented method. For this, we will use heterogeneous parallel computing platforms based on GPUs. Program parallelization aided with profiling tools would increase the maximum application performance and reduce the manual implementation efforts, thereby, offering an interesting alternative for developers to adopt the methodology.

References

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

[2] Kass M, Witkin A, Terzopoulos D (1988) Snakes: Active Contour Models. International Journal of Computer Vision 1:321, DOI https://doi.org/10.1007/BF00133570

[3] McInerney T, Terzopoulos D (1996) Deformable models in medical image analysis: a survey. Medical Image Analysis 1(2):91 – 108, DOI http://dx.doi.org/10.1016/S1361-8415(96)80007-7

[4] Pham DL, Xu C, Prince JL (2000) A survey of current methods in medical image segmentation. Annual Review of Biomedical Engineering 2:315–337, DOI 10.1146/annurev.bioeng.2.1.315

[5] Li Z, Atre R, Huda Z, Jannesari A, Wolf F (2016) Unveiling parallelization opportunities in sequential programs. Journal of Systems and Software 117:282 – 295, DOI http://dx.doi.org/10.1016/j.jss.2016.03.045

[6] Zhang Y, Tian Y (2022) A new active contour medical image segmentation method based on fractional varying-order differential. Mathematics 10(2), DOI 10.3390/math10020206

[7] Chan T, Vese L (1999) An active contour model without edges. In: Nielsen M, Johansen P, Olsen OF, Weickert J (eds) Scale-Space Theories in Computer Vision: Second International Conference, Springer Berlin Heidelberg, Berlin, Heidelberg, Lecture Notes in Computer Science, vol 1682, pp 141–151, DOI 10.1007/3-540-48236-9_13

[8] Rul S, Vandierendonck H, Bosschere KD (2010) A profile-based tool for finding pipeline parallelism in sequential programs. Parallel Computing 36(9):531 – 551, DOI http://dx.doi.org/10.1016/j.parco.2010.05.006

[9] Graham SL, Kessler PB, McKusick MK (2004) gprof: A call graph execution profiler. ACM SIGPLAN Notes 39(4):49–57, DOI 10.1145/989393.989401

[10] Dimakopoulou M, Eranian S, Koziris N, Bambos N (2016) Reliable and efficient performance monitoring in linux. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Press, Piscataway, NJ, USA, pp 1–13

[11] Rohou E, INRIA (2012) Tiptop: Hardware performance counters for the masses. In: 2012 41st International Conference on Parallel Processing Workshops, IEEE, DOI 10.1109/ICPPW.2012.58

[12] Schulz M, de Supinski BR (2007) Practical differential profiling. In: Euro-Par 2007 Parallel Processing, Lecture Notes in Computer Science, Springer, pp 97–106, DOI 10.1007/978-3-540-74466-5_12

[13] Spivey JM (2004) Fast, accurate call graph profiling. Softw Pract Exper 34(3):249–264, DOI 10.1002/spe.562

[14] Ball T, Larus JR (1994) Optimally profiling and tracing programs. ACM Transactions on Programming Languages and Systems 16(4):1319–1360, DOI 10.1145/183432.183527

[15] Shende S (1999) Profiling and tracing in Linux. In: Proc. Second Extreme Linux Workshop, USENIX Annual Technical Conference, pp 26–30

[16] Masood S, Sharif M, Masood A, Yasmin M, Raza M (2015) A survey on medical image segmentation. Current Medical Imaging Reviews 1:3–14, DOI 10.2174/157340561101150423103441

[17] Filho PPR, Cortez PC, Barros ACdS, Albuquerque VHC, Tavares JMRS (2017) Novel and powerful 3D adaptive cristp active contour method applied in the segmentation of CT lung images. Medical Image Analysis 35:503–516, DOI 10.1016/j.media.2016.09.002

[18] Oliveira RB, Filho ME, Ma Z, Papa JP, Pereira AS, Tavares JMRS (2016) Computational methods for the image segmentation of pigmented skin lesions: A review. Computer Methods and Programs in Biomedicine 131:127–141, DOI 10.1016/j.cmpb.2016.03.032

[19] Oliveira RB, Marranghello N, Pereira AS, Tavares JMRS (2016) A computational approach for detecting pigmented skin lesions in macroscopic images. Expert Systems with Applications 61:53 – 63, DOI http://dx.doi.org/10.1016/j.eswa.2016.05.017

[20] Jodas DS, Pereira AS, Tavares JMRS (2016) Lumen segmentation in magnetic resonance images of the carotid artery. Computers in Biology and Medicine 79:233 – 242, DOI http://dx.doi.org/10.1016/j.compbiomed.2016.10.021

[21] Tina, Dubey SK, Bhatt AK, Mittal M (2022) Analysis of algorithms in medical image processing. In: Tomar A, Malik H, Kumar P, Iqbal A (eds) Machine Learning, Advances in Computing, Renewable Energy and Communication, Springer Singapore, Singapore, pp 99–111

[22] Jiang D, Qu H, Zhao J, Zhao J, Hsieh MY (2021) Aggregating multi-scale contextual features from multiple stages for semantic image segmentation. Connection Science 33(3):605–622, DOI 10.1080/09540091.2020.1862059

[23] Lu H, Li Y, Wang Y, Serikawa S, Chen B, Chang J (2013) Active contour model for image segmentation: A review. In: International Conference on Industrial Applications Engineering, pp 104–111, DOI 10.12792/iciae2013.022

[24] SharifAbadi HVZMRHPHH (2020) A genetic algorithm-based tasks scheduling in multicore processors considering energy consumption. International Journal of Embedded Systems 13(3):264–273, DOI https://dx.doi.org/10.1504/IJES.2020.109957

[25] Nikov K, Nunez-Yanez J (2020) Intra and inter-core power modelling for single-isa heterogeneous processors. International Journal of Embedded Systems 12(3):324–340, DOI 10.1504/IJES.2020.107046

[26] Aitali N, Cherradi B, Abbassi AE, Bouattane O (2016) Parallel implementation of bias field correction fuzzy C-Means algorithm for image segmentation. In: (IJACSA) International Journal of Advanced Computer Science and Applications, vol 7(3), pp 375–383, DOI 10.14569/IJACSA.2016.070352

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

[28] Bader D, Jaja J, Harwood D, Davis LS (1996) Parallel algorithms for image enhancement and segmentation by region growing with an experimental study. In: Proceedings of International Conference on Parallel Processing, IEEE, DOI 10.1109/IPPS.1996.508089

[29] Gulo CASJ, Sementille AC, Tavares JMRS (2017) Techniques of medical image processing and analy-

sis accelerated by high-performance computing: a systematic literature review. Journal of Real-Time Image Processing DOI 10.1007/s11554-017-0734-z

[30] Balla-Arabé S, Gao X (2014) Geometric active curve for selective entropy optimization. Neurocomputing 139:65–76, DOI http://dx.doi.org/10.1016/j.neucom.2013.09.058

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

[32] Yang P, Song W, Zhao X, Zheng R, Qingge L (2020) An improved otsu threshold segmentation algorithm. International Journal of Computational Science and Engineering 22(1):146–153, DOI 10.1504/IJCSE.2020.107266

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

[34] Salomon M, Heitz F, Perrin GR, Armspach JP (2005) A massively parallel approach to deformable matching of 3D medical images via stochastic differential equations. Parallel Computing 31(1):45–71, DOI http://dx.doi.org/10.1016/j.parco.2004.12.003

[35] Daggett T, Greenshields I (1998) A cluster computer system for the analysis and classification of massively large biomedical image data. Computers in Biology and Medicine 28(1):47–60, DOI 10.1016/S0010-4825(97)00032-2

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

[37] Zhuge Y, Cao Y, Udupa JK, Miller RW (2011) Parallel fuzzy connected image segmentation on GPU. Medical Physics 38(7):4365–4371, DOI 10.1118/1.3599725

[38] Zhuge Y, Ciesielski KC, Udupa JK, Miller RW (2013) GPU-based relative fuzzy connectedness image segmentation. Medical Physics 40(1), DOI 10.1118/1.4769418

[39] Lamas-Rodríguez J, Heras DB, Argüello F, Kainmueller D, Zachow S, Bóo M (2016) GPU-accelerated level-set segmentation. Journal of Real-Time Image Processing 12(1):15–29, DOI 10.1007/s11554-013-0378-6

[40] Shi W, Li Y, Miao Y, Hu Y (2012) Research on the key technology of image guided surgery. Przeglad Elektrotechniczny 88(3B):29–33

[41] Saran AN, Nar F, Saran M (2014) Vessel segmentation in MRI using a variational image subtraction approach. Journal of Electrical Engineering and Computer Sciences 22(2):499–516, DOI 10.3906/elk-1206-18

[42] Weng TH, Chiu CC, Hsieh MY, Lu H, Li KC (2020) Parallelisation of practical shared sampling alpha matting with openmp. International Journal of Computational Science and Engineering 21(1):105–115, DOI 10.1504/IJCSE.2020.105217

[43] Weng TH, Chen YS (2019) On parallelisation of image dehazing with openmp. International Journal of Embedded Systems 11(4):427–439, DOI https://dx.doi.org/10.1504/IJES.2019.100859

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

[45] Carass A, Roy S, Jog A, Cuzzocreo JL, Magrath E, Gherman A, Button J, Nguyen J, Prados F, Sudre CH, Cardoso MJ, Cawley N, Ciccarelli O, Wheeler-Kingshott CA, Ourselin S, Catanese L, Deshpande H, Maurel P, Commowick O, Barillot C, Tomas-Fernandez X, Warfield SK, Vaidya S, Chunduru A, Muthuganapathy R, Krishnamurthi G, Jesson A, Arbel T, Maier O, Handels H, Iheme LO, Unay D, Jain S, Sima DM, Smeets D, Ghafoorian M, Platel B, Birenbaum A, Greenspan H, Bazin PL, Calabresi PA, Crainiceanu CM, Ellingsen LM, Reich DS, Prince JL, Pham DL (2017) Longitudinal multiple sclerosis lesion segmentation: Resource and challenge. NeuroImage 148:77 – 102, DOI https://doi.org/10.1016/j.neuroimage.2016.12.064

[46] Zou KH, Warfield SK, Bharatha A, Tempany CM, Kaus MR, Haker SJ, Wells WM, Jolesz FA, Kikinis R (2004) Statistical validation of image segmentation quality based on a spatial overlap index. Academic Radiology 11(2):178–189, DOI https://doi.org/10.1016/S1076-6332(03)00671-8

[47] Shivhare SN, Kumar N (2022) A study on brain tumor segmentation in noisy magnetic resonance images. In: Gupta G, Wang L, Yadav A, Rana P, Wang Z (eds) Proceedings of Academia-Industry Consortium for Data Science, Springer Singapore, Singapore, pp 153–166