Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Filipe de Sousa Matos Ferreira

Improving Digital Image Correlation in the TopoSEM Software Package

February 2023

Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Filipe de Sousa Matos Ferreira

Improving Digital Image Correlation in the TopoSEM Software Package

Master dissertation
Integrated Masters in Informatics Engineering

Dissertation supervised by
Alberto José Proença
André Martins Pereira

February 2023

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisors, professors Alberto Proença and André Pereira. Without their experience and extensive knowledge I would not have been able to complete this dissertation. On a more personal note, if not for their sensibility and understanding towards my personal situation during the last year it would have been impossible to balance my personal life with my academic life.

I am also grateful to my family, and especially my girlfriend, for their relentless support and for always being there to help me when my personal problems seemed nigh impossible to overcome.

Finally, I also would like to thank all my friends that were there to give nudges in the right direction.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

TopoSEM is a software package with the aim of reconstructing a 3D surface topography of a microscopic sample from a set of 2D Scanning Electron Microscopy (SEM) images. TopoSEM is also able to produce a stability report on the calibration of the SEM hardware based solely on output images.

One of the key steps in both of these workflows is the use of a Digital Image Correlation (DIC) algorithm, a no-contact imaging technique, to measure full-field displacements of an input image. A novel DIC implementation fine-tuned for 3D reconstructions was originally developed in MATLAB to satisfy the feature requirement of this project. However, near real-time usability of the TopoSEM is paramount for its users, and the main barrier towards this goal is the under-performing DIC implementation.

This dissertation work ported the original MATLAB implementation of TopoSEM to sequential C++ and its performance was further optimised: (i) to improve memory accesses, (ii) to explore the available vector extensions in each core of current multiprocessor chips processors to perform computationally intensive operations on vectors and matrices of single and double-precision floating point values, and (iii) to additionally improve the execution performance through parallelization on multi-core devices, by using multiple threads with a front wave propagation scheduler.

The initial MATLAB implementation took 3279.4 seconds to compute the full-field displacement of a 2576 pixels by 2086 pixels image on a quad-core laptop. With all added improvements, the new parallel C++ version on the same laptop lowered the execution time to 1.52 seconds, achieving an overall speedup of 2158.

KEYWORDS Digital Image Correlation, Scanning Electron Microscope, 3D reconstruction

RESUMO

TopoSEM é um programa cujo objetivo é reconstruir em 3D a topografia de uma amostra capturada por um microscópio electrónico de varrimento. Esta ferramenta é também capaz de gerar um relatório sobre a estabilidade da calibração do microscópio com base apenas em imagens capturadas.

Um dos passos chave para ambas as funcionalidades trata-se da utilização de um algoritmo de Correlação Digital de Imagens (DIC), uma técnica de visão por computador que não envolve contacto direto com a amostra e que permite medir deslocamentos e deformações entre imagens. Criou-se uma nova implementação de DIC em MATLAB especialmente formulada para reconstrução 3D. No entanto, a capacidade de utilizar o TopoSEM em quase tempo real é fundamental para os seus utilizadores e a principal barreira para tal são os elevados tempos de execução da implementação em MATLAB.

Esta dissertação portou o código de MATLAB para código sequencial em C++ e a sua performance foi melhorada: (i) para otimizar acessos a memória, (ii) para explorar extensões de vetorização disponíveis em hardware moderno para otimizar operações sobre vetores e matrizes, e (iii) para através de paralelização em dispositivos multi-core melhorar ainda mais a performance utilizando para isso vários fios de execução com um escalonador de propagação em onda.

A implementação inicial em MATLAB demorava 3279.4 segundos para computar uma imagem com resolução de 2576 pixels por 2086 pixels num portátil quad-core. Com todas as melhorias de performance, a nova implementação paralela em C++ reduziu o tempo de execução para 1.52 segundos para as mesmas imagens no mesmo computador, atingindo um *speedup* de 2158.

PALAVRAS-CHAVE Correlação de Imagens Digitais, Microscópio Eletrônico de Varrimento, Reconstrução 3D

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

Digital Image Correlation (DIC) encompasses a wide set of techniques to measure changes in 2D and 3D images. These techniques, originally employed in the 1980s in the field of mechanical engineering, have recently increased in popularity for micro- and nano-scale testing of deformations in structures, due to their ease of use and accuracy of the results. The output of this type of analysis is a set of displacements that measure the correlation of each subset of a reference image (for instance, a pixel) with a deformed image, for each dimension.

The TopoSEM software package reconstructs the 3D surface topography of a microscopic sample from a set of 2D Scanning Electron Microscopy (SEM) images, through the use of state-of-the-art image visualisation algorithms and patented methods. A Scanning Electron Microscope is a type of microscope that, contrary to a conventional devices that use light for information capturing, uses a focused beam of accelerated electrons that scans the sample in a raster pattern. When this beam hits the atoms on the surface of the sample it loses energy as heat and radiation. This energy loss can be measured, processed and analysed to generate an image of the sample surface. Since the wavelength of an electron is much smaller than the wavelength of a photon from a beam of light, it is possible to capture much more intricate detail with this kind of instruments.

TopoSEM provides several advantages over its competition, namely: (i) no need for a time consuming calibration of the SEM device, which is usually performed every time the sample to be observed is changed; (ii) all calculations are software-based, no need of hardware modifications nor re-scanning of the entity, as is the case of competing solutions; (iii) no direct input from the SEM device (thus, TopoSEM can be used on- and off-site, on personal computers and server environments); and (iv) the topography reconstruction can be accurately performed with only three SEM images of a sample, captured at slightly different angles. This package has been developed by Electron Softview, a recently created spin-off that will transfer TopoSEM to the market before the end of this year.

TopoSEM is composed by two independent packages that work together to analyse the input SEM images and compute the 3D topography, the DIC and the TCore tools, respectively. The former is responsible for analysing the input SEM images to calculate a set of displacements, i.e., differences between the same features in the input images, considering the rigid body physics of the sample and the optical properties of the SEM device. The latter computes the 3D topology of the recorded sample based on the displacement information calculated by the DIC tool, using patented methods. TCore is also capable of analysing the quality of the calibration of the SEM device and warn the user, based on the DIC information of two SEM images, recorded with similar hardware configurations, using the algorithms described in Mansilla et al. (2014).

From this description it is clear that both TopoSEM workflows, the 3D topography reconstruction of a sample and the numerical stability analysis of the SEM device calibration, depend on the calculation of the image dis-

placements by the DIC tool. The DIC workflow is subdivided into an initial feature matching between input SEM images, either fully automated or resorting to user input, and the calculation of the full-field displacement of the images, which can include translations, rotations, shear, and scale changes. While DIC algorithms have been extensively studied, mostly in terms of their numerical stability and accuracy, their execution time accounts for the majority of the overall time of an analysis in TopoSEM, especially if all types of distortions are considered.

TopoSEM integrates a DIC tool designed for the specific requirements of the 3D topographic reconstruction and the calibration analysis, to address the lack of features and difficulty of extending and/or improving the algorithms in existing libraries. However, the performance of this fine-tuned implementation, which was developed in MATLAB, is the main limitation to the near real-time performance requirements expected from TopoSEM users.

1.1 CHALLENGES AND GOALS

When a researcher requests time on a SEM microscope there is a limited time frame to work with due to the high operational cost of these devices. This time must be spent on obtaining images from samples and not waiting for computations from the TopoSEM software package. Thus, in this environment, 3D image reconstructions must be performed in near real-time to maximise the usage of the SEM device.

As previously stated, preliminary measurements showed that the current DIC algorithm implementation in MATLAB is the main bottleneck in achieving near real-time computation. Although the main design goal of MATLAB is simplifying operations on vector and matrices, this programming language is slow performing said operations when compared with other languages that are tailored towards high performance computing (HPC). Since the DIC algorithm can be characterised by a set of computationally intensive operations on vectors and matrices of single- and double-precision floating point values, MATLAB is non-ideal when the main goal is an efficient code capable of near real-time computation..

Therefore, the main goal of this work is to create a standalone DIC implementation capable of near real-time response. It should be achieved by porting the currently existing MATLAB implementation, to a more performant programming language, in this case C++. The new implementation explores several sequential optimisations, such as vectorization. Furthermore the code was optimised, through parallelization, with the aim of exploring the performance available in shared memory servers with one or more multi-core Processing Units (PU).

The new implementation must fulfil the functional requirements to be able to later be integrated into the TopoSEM software package. Furthermore it must maximise the usage of the computational resources available in (i) personal computers, either desktops or laptops with limited computing resources, where the goal is real-time processing of a single 3D reconstruction; and (ii) on future versions of the TopoSEM certified to work on high-performance unix servers, where the goal is improved throughput for processing thousands of images.

1.2 DOCUMENT STRUCTURE

The document is structured as follows: Chapter 2 presents an overview of the state-of-the-art algorithms and qualitative analysis of the frameworks in DIC, as well as a contextualisation of the optimisations that were explored. Chapter 3 describes the workflow in the TopoSEM application and presents the current MATLAB implementation and its limitations. It also discusses the steps taken in this work to improve the performance of the DIC library

in the TopoSEM, detailing its efficiency requirements, and how this port was validated. Chapter 4 describes the testbeds and how they were used to obtain performance metrics. In addition, the performance results will be analysed and discussed. Chapter 5 presents an overview of the work done and future work.

2

STATE OF THE ART

In this chapter the state of the art of the technologies used in this dissertation will be explored. Firstly, Section 2.1 will present the evolution, state of the art and usecases of Digital Image Correlation (DIC) technologies. Then the currently available hardware architectures will be reviewed in Section 2.2. The parallelisation strategies and technologies will be evaluated and their limitations discussed in Section 2.3 and Section 2.4. Lastly, Section 2.5 presents extra tools used for development and benchmarking of this dissertation.

2.1 DIGITAL IMAGE CORRELATION

The DIC algorithm consists of computing the alterations and deformations (translation, rotation, and scaling) that occurred between two images represented in the form of a Displacement Field. This is a structure that contains Displacement Vectors that represent 2D deformations for each grid point. Each displacement vector contains six floating point numeric values used to represent the deformation that occurred for a given grid point. A grid point is defined as a rectangular subset of the original image (Figure 1).

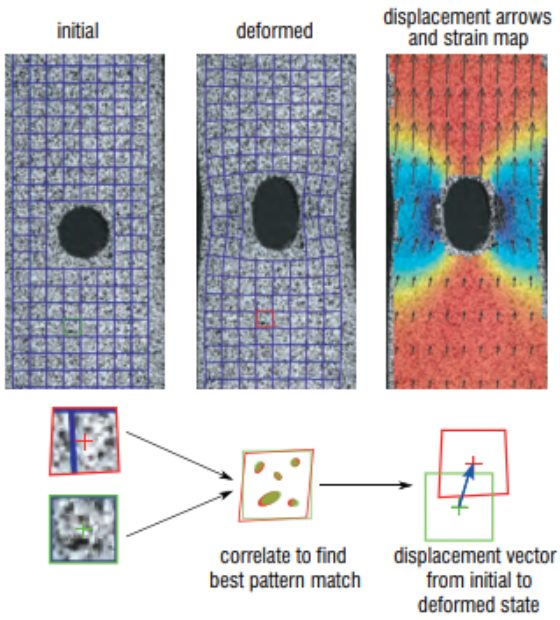


Figure 1: Simplified diagram of the DIC workflow (from Analytical Technologies (2020))

One of the methods for calculating the DIC consists on finding the maximum of the correlation array between two image subsets. The two-dimensional discrete cross correlation between a subset of the original image and a subset of the deformed image, when translated by the vector (i, j), can be defined by Equation (1).

$$r_{ij} = \frac{\sum_m \sum_n [f(m+i, n+j) - \bar{f}][g(m,n) - \bar{g}]}{\sqrt{\sum_m \sum_n [f(m,n) - \bar{f}]^2 \sum_m \sum_n [g(m,n) - \bar{g}]^2}} \tag{1}$$

where:

- $f(m, n)$ is the pixel value at the point (m, n) of the original image.
- $g(m, n)$ is the pixel value at the point (m, n) of the deformed image.
- $\bar{f}$ and $\bar{g}$ are mean values of the intensity matrices f and g respectively.

By computing the maximum for the function r_{ij} the coordinates of the point give the translational shift that occurred between the two images for a given point. To include more deformations than only translations there is a need to expand the formula by computing the relation between the point in the original image (x, y) and the deformed image (x^*, y^*) with:

$$\begin{aligned} x^* &= x + u + \frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y \\ y^* &= y + v + \frac{\partial v}{\partial x}\Delta x + \frac{\partial v}{\partial y}\Delta y \end{aligned} \tag{2}$$

The elements of the Displacement Vector are based on the pair of formulas from Equation (2) and are as follow:

$$(u, v, \frac{\partial u}{\partial x}\Delta, \frac{\partial u}{\partial y}\Delta, \frac{\partial v}{\partial x}\Delta, \frac{\partial v}{\partial y}\Delta)$$

However, in practical applications, the correlation array is usually computed using the Fast Fourier Transform (FFT) since it is comparatively much faster. An implementation of a DIC algorithm using this method was first proposed by Anuta (1970). Although this approach is computationally efficient, the accuracy of the results produced is subpar. Thus, future research focused mainly on improving the quality of the resulting displacement field.

Sutton et al. (1986) first proposed the usage of the Newton-Raphson (NR) algorithm. By starting with an initial guess that is close to the final result, this algorithm produces successively better approximations with each iteration. Therefore, the higher the number of iterations performed, the better the final result will be. One of the methods for obtaining the initial guess for this algorithm is by using the FFT algorithm previously described as the initial guess for the NR algorithm for a predefined point of the image. Then each subsequent grid point uses the displacement of a neighbouring grid point that has already been computed as the initial guess.

In a iterative algorithm there is a need to quantify the progress that is being made. In the case of the DIC algorithm this is quantified by how similar the current subset on the reference image is to the subset on the target image when the current displacement is applied. One of the algorithms that can be used for this comparison is the Zero-Normalised Sum of Square Differences ($ZNSSD$) (Equation (3)), Pan et al. (2010). The result of this

equation is closer to 0 if the two subsets are more similar and the value increases as the subsets become more distinct.

$$ZNSSD = \sum_m \sum_n \left(\frac{f(m,n) - \bar{f}}{\Delta f} - \frac{g(m,n) - \bar{g}}{\Delta g}\right)^2 \tag{3}$$

where:

- f and g are the reference subset and the target subset respectively.
- $f(m,n)$ and $g(m,n)$ are the pixel value at the point (m,n) on the subset f and g respectively.
- $\bar{f}$ and $\bar{g}$ are the average of all pixel values in the corresponding subset.
- Δf and Δg are the delta of the corresponding subset (computed using Equation (4)).

$$\Delta f = \sqrt{\sum_m \sum_n (f(m,n) - \bar{f})^2} \tag{4}$$

where:

- f is a image subset.
- $f(m,n)$ is the pixel value at the point (m,n) on the subset f
- $\Delta f$is the delta of the subset f
- $\bar{f}$ is the average of all pixel values on the subset f

Pan et al. (2013) achieved a significant speedup by replacing the NR algorithm with the Inverse Compositional Gauss-Newton algorithm (IC-GN). This, similar to NR, is also an iterative algorithm that, based on an initial estimation, refines the results with each iteration. Yet, this algorithm converges faster towards the real solution, thus performing less iterations with each computation, making it much faster than NR.

By applying an interpolation algorithm on the input of the DIC it is possible to achieve sub pixel precision. This class of algorithms consists of obtaining new data points from a discrete set of known data points. When applying an interpolation algorithm to the pixels of the image it is possible to interpolate values between the pixels, which allows the DIC algorithm to achieve sub pixel precision. Thereby increasing the accuracy of the full field displacement. The most common interpolation algorithms are Bi-linear, Bi-cubic or Bi-spline. Schreier et al. (2000) shows that the usage of a Bi-cubic interpolation algorithm greatly reduces error introduced by the interpolation when compared to a Bi-linear interpolation algorithm. When comparing a Bi-cubic algorithm with a Bi-Quintic, or higher, the increase in computation cost is much higher than the increase in result quality. Therefore, the interpolation algorithm most commonly used for DIC computation is the Bi-Cubic since it presents a balance between computation time and result quality.

Couty et al. (2021) proposed the use of image pyramids to speedup the DIC computation under certain scenarios. This method consists in reducing the resolution of the images by a factor of 32 and applying the IC-GN algorithm. Once the result for that level is achieved, the original image is reduced by 16 times and the result of the previous layer is used as initial guess for the new layer. This cycle of using the previous layer as

initial guess to the new layer is repeated until the values for the original resolution are calculated. The closer the initial guess the faster the DIC computation will be. The initial guess for the first layer is the farthest from the real solution for that given layer. The subsequent layers use the previous layer as initial guess, making it closer to the real solution. Therefore, under some scenarios the cost of computing several layers can be offset by the benefits that come from using a better initial guess.

While the current TopoSEM implementation relies on the user for the initial guess estimation for the IC-GN algorithm, some DIC algorithms obtain this value by analysing the provided input images. The method that is currently the most used borrows from the first version of DIC algorithms, and use the output of the FFT algorithm as input for the iterative algorithm that refines the result. Others, such as Thiagu et al. (2020), use pyramid methods to approximate the value.

2.1.1 *Use cases of the DIC algorithm*

Although the aim of this dissertation is developing an efficient DIC algorithm that outputs displacements and additional information to be used in a 3D reconstruction, this is not the only use case. In fact, there are many scientific areas that can benefit from the usage of this algorithm.

Puymbroeck et al. (2000) applied the DIC algorithm to images captured from optical satellites with the aim of measuring distortions caused by earthquakes. The traditional techniques that are used to measure this deformations are usually limited to a predefined small area. This is not ideal since the dimension of the affected area is not know beforehand. By applying a DIC algorithm with sub-pixel precision to a before and after image captured via an optical satellite of the general area, it is possible to compute the deformation that occurred in large areas with minimal error. Thus, allowing the in-depth study of the large scale effects of an earthquake that otherwise could not be perceived.

Moisy et al. (2008) researched the use of DIC in the field of fluid dynamics to measure disruptions and currents on liquid interfaces with a high level of success. This was achieved by drawing a grid of equidistant dots just below the liquid surface and comparing the before and after picture of the experimental setup. The light refraction alters the perceived position of the dots, and that refraction is altered by disruptions on the liquid. Therefore, it is possible to compute the alterations on the dot's position via the DIC algorithm and infer the disruption that caused it.

Zhong et al. (2018) explored the concept of using a DIC algorithm to aid in the generation of 3D movies. By applying a DIC algorithm to the video captured by two side-by-side cameras pointed in parallel at the same scene it is possible to generate data that can later be used to compute depth information. In order to achieve real-time performance some optimisations for this specific usecase were developed. Namely, since both images were captured simultaneously and side-by-side it is possible to optimise the DIC algorithm to explore the parallax effect. When viewing the same scene from two distinct points of view the observed objects appear to have different positions for each observer. The apparent difference in position between observers tends to have the same direction as the vector between the two points of view. Since the relative position of the two cameras remain the same during the recording, the full field displacement tends to have the same direction across the multiple computations and inside each frame. This allows for the iterative algorithm to use the DIC results of previous frames and displacement of previous grid points in the current frame as a initial guess that is much

closer to the final value for the current grid point. Hence greatly accelerating the DIC computation for this specific situation.

As it is possible to see from these examples, the development of an efficient and expansible DIC algorithm is paramount for the advancement in many fields.

2.1.2 *Tools for DIC computation*

DIC tools for the analysis of a variety of images are widely available in the scientific and industrial communities. Different tools often have different purposes, which impact their design, usability, and accuracy on different scenarios. Some relevant tools currently available are described in the following subsections.

NCorr

NCorr[1], created by Blaber et al. (2015), is an open source 2D Digital Image Correlation algorithm implemented in MATLAB. To mitigate the negative impact that MATLAB has on the performance of a program the areas that are more computationally intensive where optimised mainly through the usage of C++/MEX. A MEX file is a MATLAB functionality that allows for the program to call C, C++ and FORTRAN functions. In the case of NCorr the MEX files are using C++.

This DIC algorithm uses Fast Normalised Cross Correlation (FNCC) to compute the initial guess for the iterative algorithm. FNCC is derived from the Cross Correlation algorithm described in Section 2.1. The Gauss-Newton iterative algorithm is used to compute the deformation at each point. In order to achieve sub pixel precision, NCorr uses the Biquintic B-spline interpolation algorithm.

This program was designed to be used as a standalone Graphical User interface (GUI). Although this makes the usage of the program easier for obtaining DIC results, it becomes harder to integrate as a library. Furthermore, the top level API is designed in MATLAB incurring in the same issues of the novel MATLAB implementation created for TopoSEM. The increased complexity also makes it harder to extend the functionalities of the current NCorr implementation. The aforementioned limitations eliminate this library as a candidate to replace the current MATLAB implementation.

DICe

DICe[2] (Digital Image Correlation Engine), Turner (2015), is an open source DIC tool programmed in C++ and designed to be used as a library in another program or as a standalone tool. DICe offers two main functionalities using the DIC algorithms: full field displacement computation and object tracking. Both can take advantage of MPI parallelism as well as on-core parallelism.

One of the main differentiating feature of this tool is the ability of defining irregularly shaped subsets. This enables and improves the tracking of oblong objects but offers little to no advantage for full field displacement computation.

1 NCorr is available for download at http://www.ncorr.com/
2 DICe is available for download at https://github.com/dicengine/dice

GPUCorrel

GPUCorrel[3] , Couty et al. (2021), is a Python DIC library implementation that uses GPU acceleration and is one of the first examples of a tool that implements the pyramid optimisation described in the Section 2.1 to compute DIC. It starts with a image with the size reduced by a factor of 32. The scaling is performed using the texture mapping units in the GPU. Then, it creates a new image with size reduced by a factor 16 (half of the previous step) and uses the computed full-field displacement from the previous step as initial guess for the current image. The process is repeated until the full field displacement for the original image is computed.

In some situations, the advantage of using better initial guesses for each layer offsets the cost of applying the DIC algorithm more times. Although the concept can result in a speedup in some niche situations this tool does not support sub pixel precision. Furthermore, the tool itself has poor error handling and reporting, the API does not expose all the needed features, and the tool is programmed in python making it incompatible with the rest of the TopoSEM software package and eliminating it as a candidate.

2.2 HARDWARE ARCHITECTURES

As previously stated, the resulting tool must run efficiently both on personal laptops and homogeneous servers. These computing environments have different architectures and it is necessary to understand their differences in order to explore their computing capability. While personal laptops only contain one Processing Unit (PU) (referred to as single-socket), homogeneous servers can contain a single or multiple multi-core PU of the exact same make and model (referred to as multi-socket).

In a dual-socket machine (Figure 2), i.e. a computer with two PU, each PU has its own memory hierarchy. Most of the currently available PUs contain a memory hierarchy consisting of L1, L2, L3 and RAM, and a high bandwidth interface that allows the communication between the two PUs. The interface is used to share the memory hierarchy address space between the PUs in a Non-Unified Memory Architecture (NUMA). This means that the memory access time depends on the memory location in relation to the processor. For example, a address in RAM that is mapped on one PU takes more time to be accessed from the other PU because it needs to pass trough the communication interface. Multi-socket architectures introduce some limitations to the computation. Due to the limited bandwidth between PUs it is possible to saturate this interface and incur in a bottleneck. The increased latency of accessing memory connected to another PU also negatively impacts computing performance because of thread starving. To mitigate this issue, a multi-threaded program that executes in a multi-socket environment must distribute its threads in a way that each thread executes in the PU that is directly associated with the memory that it uses. These limitations do not exist on single-socket environment since there is only one PU and one memory hierarchy.

3 GPUCorrel is available for download at https://github.com/Dad0u/GPUCorrel

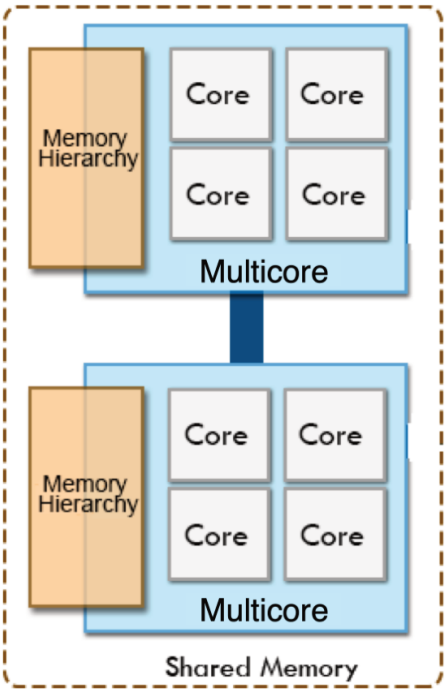


Figure 2: Representation of a dual-socket homogeneous server with two multi-core PU devices (from Pereira (2019))

The clock speed and memory bandwidth needs of PUs have historically increased much faster than DRAM speed (Wulf and McKee (1995) and Carvalho (2002)). To mitigate this issue modern PUs use the aforementioned memory hierarchy. Each level closer to the core represents a significantly faster memory with smaller storage size. This is caused by the higher complexity and costs associated with faster cache memory. Code can be optimised to explore this behaviour by using cache locality. If the program accesses memory addresses that are close together it is more likely that the memory address is already on cache memory, reducing memory access latency.

Modern PUs support both Single Instruction Single Data (SISD) and Single Instruction Multiple Data (SIMD). SISD consist of having an instructions that only affects a single piece of data. SIMD consists of a instruction that can operate over a vector of data simultaneously. Although SIMD has the potential of being considerably faster than SISD, there are several drawbacks. First and foremost, not all programs can directly take advantage of SIMD instructions. Thus, there is a need to spend more time and resources on creating a program that explores SIMD. Secondly, expanding the instruction set to include SIMD instructions, adding vector registers and expanding the Arithmetic logic units (ALU) introduces higher hardware complexity and cost. The two most comonly used extensions of the x86 Instruction Set Architecture (ISA) to support SIMD: AVX-256 and AVX-512. These extensions allow performing operations on vector registers of 256 and 512 bits respectively.

The initial focus of hardware development was on increasing single core performance instead of increasing core count. A wall on single core performance improvement was reached around 2002 with the release of the Intel Pentium 4 (Figure 3). This lead to a change in focus to increase the core count. For example, recently AMD released the AMD EPYC 9654, a 96 core PU (AMD (2022)).

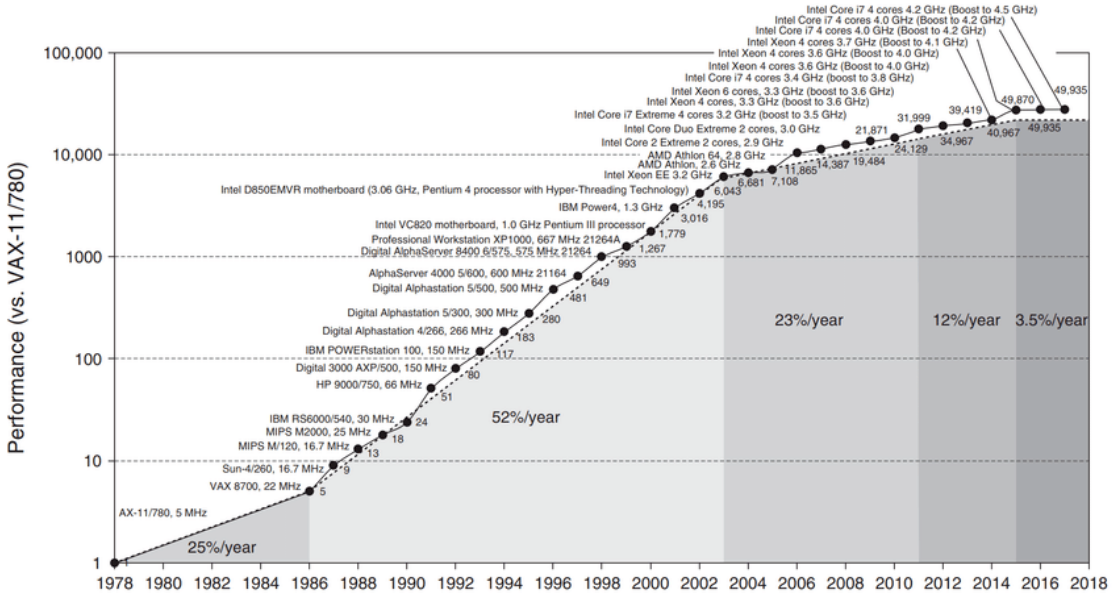


Figure 3: Growth in PU Performance over 40 years (from Hennessy and Patterson (2018))

2.3 PARALLELISM

Several APIs and libraries with the aim of aiding in the development of parallel applications are already available. These tools aim to simplify the process of creating a straightforward parallel application. However, they limit the way the developer can manage the behaviour of the parallel workers. Instead of using a pre-exisiting tool for parallelism, a more bare-bones approach is to create and manage the parallelism from scratch. Although it increases the development complexity and time, it is possible to tailor a solution to a specific problem. A solution using this method is no longer constrained by the architecture, features and design decisions of one specific pre-existing parallelisation tool.

In the following subsections the most commonly used libraries and tools for the creation of parallel applications will be discussed and compared.

OpenMP

OpenMP (Open Multi-Processing), Dagum and Menon (1998), is a set of multi-platform compiler directives that implement a shared-memory multiprocessing API for C, C++ and Fortran. Since it is implemented by the compiler it is widely available on many platforms, instruction-set architectures and Operating Systems ,including Linux, MacOS and Windows. The main goal of this API is providing a simple interface for the development of simple parallel programs with scalable performance.

Its scheduler is based on a work sharing strategy, where a master thread spawns a set of worker threads to simultaneously compute a task, or different tasks, on a shared data structure. This approach is also efficient for irregular workloads. Currently OpenMP supports the scheduling types: static, dynamic and guided.

OpenMPI

Open Message Passing Interface (OpenMPI) library, Boku (2004), was created with the aim of providing a simple yet powerful process-based programming in distributed memory environments. This tool is available for C, C++ and Fortran.

Unlike OpenMP, OpenMPI has no pre-implemented scheduling strategy. It is up to the developer to implement data and task scheduling strategies, explicitly code the communication of data and tasks between nodes, and to create and manage an adequate amount of processes for the applications and servers used. Therefore, this is a tool that requires high level of expertise to develop efficient parallelisation strategies for complex applications.

OpenMPI is often used in conjugation with a shared memory parallel programming API, such as OpenMP. Where OpenMPI is responsible for orchestrating the different nodes in the computing environment and OpenMP explores efficient parallel execution inside each node.

pthreads

POSIX Threads (*pthreads*) consists of a library for C and C++ for thread management, MUTEX, condition variables and synchronisation between threads. Although it was initially developed for Unix systems there are solutions to run it on Windows devices with little to no modification.

With the usage of the *pthreads* library it is possible to manually introduce parallelisation in a program. This library allows for the creation and management of worker threads and the management of the data structures. Thus, by defining a scheduling strategy managed by a main thread it is possible to create a shared-memory multiprocessing application that is not limited by the architecture of any tool, such as OpenMP.

2.4 SCHEDULING

Parallel task scheduling is an optimisation strategy where a section of the computation is divided in jobs and distributed amongst the available workers to be computed simultaneously. The jobs can be codependent and take different times to compute. After the jobs have been defined they can be bundled in sets of one or more jobs, usually called chunks or batches.

In theory, by dividing work in a set of chunks and efficiently distributing them by N workers it would lead to an N fold improvement of the computation time. This indicates that the relation between computation time and number of workers should be linear. However, in reality there is often a close-to-linear improvement for a small number of workers, which flattens out into a constant for larger values. Amdahl (1967) first proposed an explanation for this behaviour: Since there is a section of the program that benefits from the performance optimisation and a section that does not, when the number of workers increases and the computation time of the parallelizable part decreases the computation time of the sequential section remains the same. The sequential part can be, for example, the data initialisation in the beginning of the program.

This means that the optimised program can never be faster than the sequential component and will, eventually, reach a plateau on the execution time improvement for a large number of workers. To mimic this behaviour, a mathematical formula known as *Amdahl's law* was proposed:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}} \tag{5}$$

where:

- S_{latency} is the theoretical speedup of the whole task;
- s is the speedup of the part that benefits from performance optimisation;
- p is the proportion of execution time that the part benefiting from performance optimisation previously occupied.

Furthermore, the distribution strategy of the chunks needs to be tailored to the current problem. If there are more workers than there are threads in the PU it can lead to resource starving and reduction in the performance of the algorithm. If the work does not have enough granularity to be equally divided among the workers it can lead to worker starving. This either means that while some workers are executing jobs, other workers are waiting for chunks for a extended period of time or that some workers finish faster than others.

The simplest kind of scheduler is the **Static** Scheduler where the workload is equally divided amongst each worker previous to the parallel section and the number of elements of each chunk is statically defined by the user. Static scheduling has the smallest overhead of all scheduler types since there is no need to compute the work distribution during the parallel section. This type of scheduler can easily incur in thread starving if the computation time of the total work assigned to a worker is smaller than the one assigned to others. To attenuate this issue, **Dynamic** Scheduling assigns work, that is divided into static user-defined chunk size, to each worker based on their throughput. More work is assigned to a worker only when that worker finishes the current work. Thus, this method allows to more evenly distribute and balance the computation of the total work between the workers when the work size is variable. However, this also means that there is more overhead than in the static scheduler. To give even more granular control to the scheduler, **Guided** Scheduling allows the control of the chunk size during runtime. Similar to dynamic scheduling, it divides the chunks based on each threads throughput. However, the size of the chunks is adjusted based on the number of remaining chunks divided by the number of workers, until a user-defined minimum chunk size is achieved. This can allow for less overhead when compared to dynamic schedulers if explored correctly.

EFT (Earliest Finish Time) Scheduling (Zhao and Sakellariou (2003)) is designed to distribute a set of codependent jobs amongst workers in a homogeneous computing environment. The codependent jobs are represented as a directed acyclic graph where the weighted vertices are the jobs, typically with the weight value being the computation cost, and the weighted edges represent the dependency between jobs, typically weighted with the communication cost of the task. Based on the information of the dependency between jobs and the computation and communication cost of each job, the scheduler first assigns a priority value to each job and then distributes them across the heterogeneous workers, starting with the highest priority. This algorithm can produce near optimal scheduling. In spite of that, for more complex computations it can have a negative impact in the computation time. This is due to the fact that this is a greedy algorithm that has no context on how the order of the job being currently chosen will impact the execution of its children. **HEFT** (Heterogeneous Earliest Finish Time) Scheduling (Topcuoglu et al. (2002) and Radulescu and van Gemund (2000)) is derived from the EFT scheduler but instead of having a homogeneous computing environment it is able to efficiently distribute the tasks across

heterogeneous workers. There are some variants of the HEFT scheduler, such as Bittencourt et al. (2010), that attempt to mitigate the issues that come from the greedy nature of the algorithm by performing a look-ahead to analyse how the current decision will impact the overall computation.

In spite of the great speedups that can be achieved, the focus on parallelization is relatively recent. As stated in the Section 2.2, due to technical difficulties the main focus of hardware development prior to 2002 was on increasing the single core performance without being able to increase the core count. As a result of the low core count on commercially available PUs, the main focus of software development was on exploring sequential performance, relegating parallel computing to niche academic research. The excessive focus on sequential computing at that time even lead Torvalds (2001), the creator of the Linux Kernel, to state that "*the current [Linux Kernel] scheduler has the same old basic structure that it did almost 10 years ago, and yes, it's not optimal, but there really aren't that many real-world loads where people really care.*".

However, around 2002, a limit was reached on how fast single core performance could improve, leading to a renewed focus on increasing core count on available PUs. The increase in core count revived the focus on parallel computing in real world applications. One example of this mentality shift is the Linux Kernel. Around 2001 the Linux Kernel Scheduler was considered completed. Some going as far as stating that scheduling tasks was a solved issue and there was no need to further research this field of HPC. Nevertheless, Lozi et al. (2016) found that with the evolution of the available hardware, the Linux Kernel Scheduler left a lot of performance on the table. In fact, there were many instances where there were tasks waiting on a queue to be processed while there were workers waiting for a significant amount of time for a new task to process. This inefficient scheduling lead to a 14-23% decrease in performance on popular databases, illustrating the need of carefully building schedulers tailored towards fully exploring the modern PUs capabilities.

2.5 OTHER TOOLS AND LIBRARIES

Some other tools were used in the development of this dissertation.

Open Source Computer Vision library (**OpenCV**) is a library aimed at real-time computer vision and includes tools for image loading and manipulation and tools for matrix operations. It is widely available for many platforms and architectures, including Windows and Linux, and is written in C++ but bindings are available for Python, Java and MATLAB. OpenCV also supports hardware acceleration using CUDA and OpenCL.

vTune is a tool developed by Intel with the aim of performing performance analysis on x86 systems. It is available on both Windows and Linux and supports many languages including C, C++ and FORTRAN. This tool provides many different profiling methods including hotspots, memory consumption and threading. The hotspot profiling shows which function calls consume more computation time. By targeting this areas it is possible to focus and maximise the optimisation efforts.

3

THE TOPOSEM SOFTWARE PACKAGE

The TopoSEM software package is a tool designed to be used with Scanning Electron Microscopy (SEM) images to achieve two distinct computational workflows: the topographical reconstruction and the analysis of the SEM device measurement stability. The former workflow consists of analysing three distinct images (A, B, C) and generating a 3D reconstruction of the samples surface. One of the steps of the pipeline for this workflow consists of computing two full field displacements $(A + B, B + C)$ with a Digital Image Correlation (DIC) algorithm. The latter workflow is used to identify possible issues with a SEM calibration. One of the steps of the pipeline of this workflow also involves the computation of a full field displacement.

To compute the displacements, a novel DIC this algorithm was created. This implementation uses the IC-GN algorithm coupled with the Bi-Cubic interpolation. The initial guess has to be provided to the algorithm and before being used passes a step of brute-force refinement that checks in a 3 by 3 area of the initial guess if there is any better possible input.

3.1 USING TOPOSEM

There are two distinct workflows in the TopoSEM software package. The first one allows the user to create a 3D reconstruction of the scanned samples, and the second one allows the user to perform a stability analysis of the SEM device calibration. This section presents a summarised description of these workflows from an user's perspective. An in-depth description of the usability of TopoSEM is available in Appendix A.

The workflows supported by TopoSEM require the user to provide a set of three images of a sample, captured at slightly different angles. There is no need to know the magnitude of the tilting to perform an analysis with TopoSEM, as the tool is capable of inferring the angles if an accurate calculation of the displacements is performed.

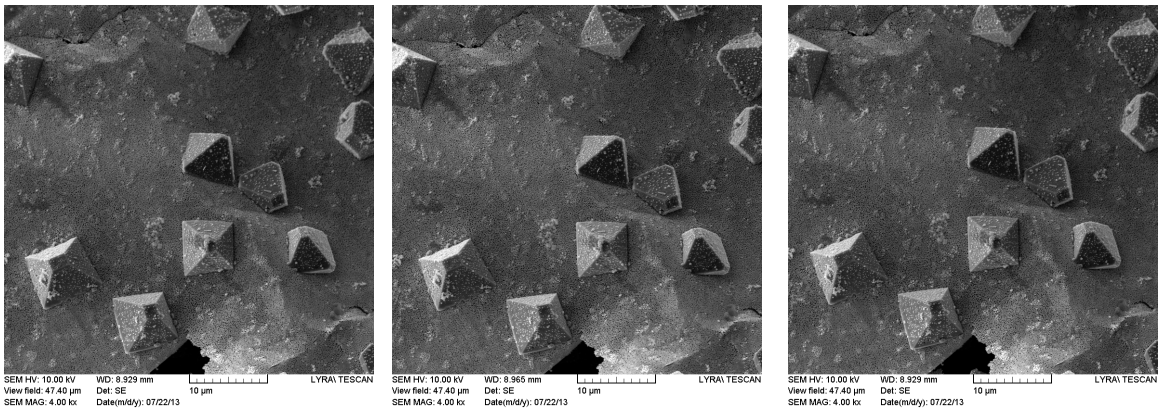


Figure 4: Sample SEM images used in this workflow demonstration.

Apart from the reference image and the target image the user can configure more parameters of the DIC computation. The Figure 5 represents the parameters related to the generation of the grid of subsets, which are compared between images by the DIC algorithm. The user can change the size of the sides of the subsets (in blue) and the vertical and horizontal spacing between the centre point of each subset (in purple). Depending on the values chosen there might be a border around the image that is ignored for the generation of the subsets on the reference image (in grey). The user can also configure the convergence criterion and the maximum number of iterations for the IC-GN algorithm.

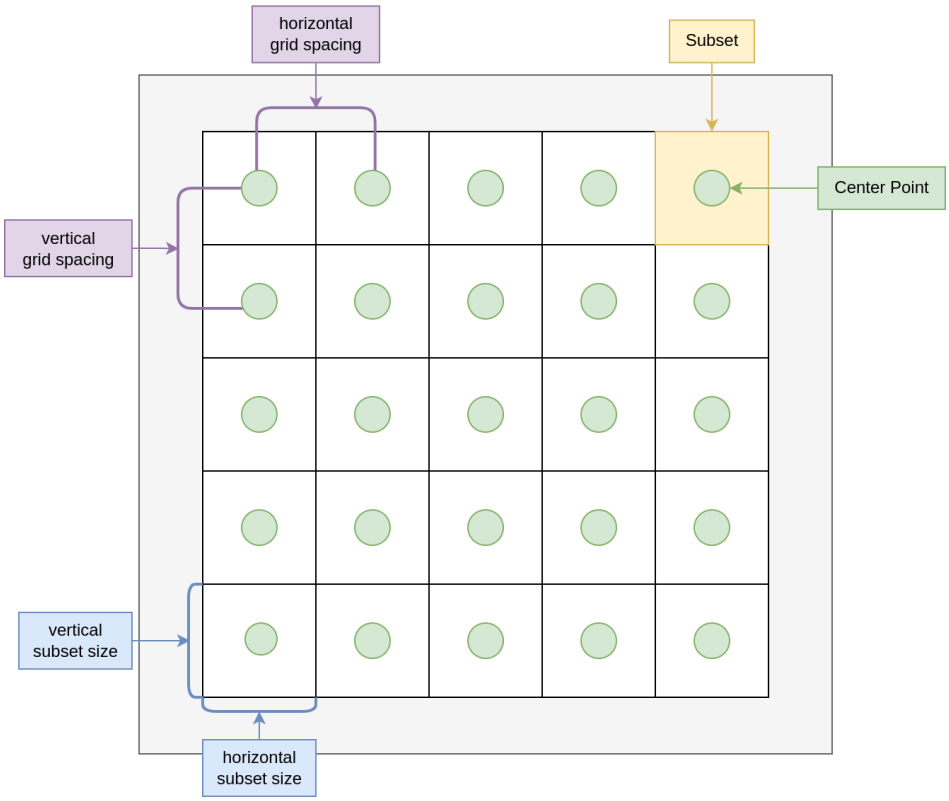


Figure 5: Configuration of the DIC subsets.

3.1.1 *Topographical 3D Reconstruction*

The **Analysis menu** allows the user to load a set of three SEM images taken from the same sample at different view points, such as the images presented in Figure 4. Afterwards the user must input the SEM's Working Distance (WD) and define the scale of the image (i.e., the physical distance corresponding to the size of a pixel), both are present on the footer of the image. The user can also define a Region of Interest (ROI) on the image. Figure 6 shows the main screen of the topographical analysis, where the list of input images can be seen on the top left, and the configuration parameters on the bottom, categorised as configuring the DIC computation and the following 3D reconstruction.

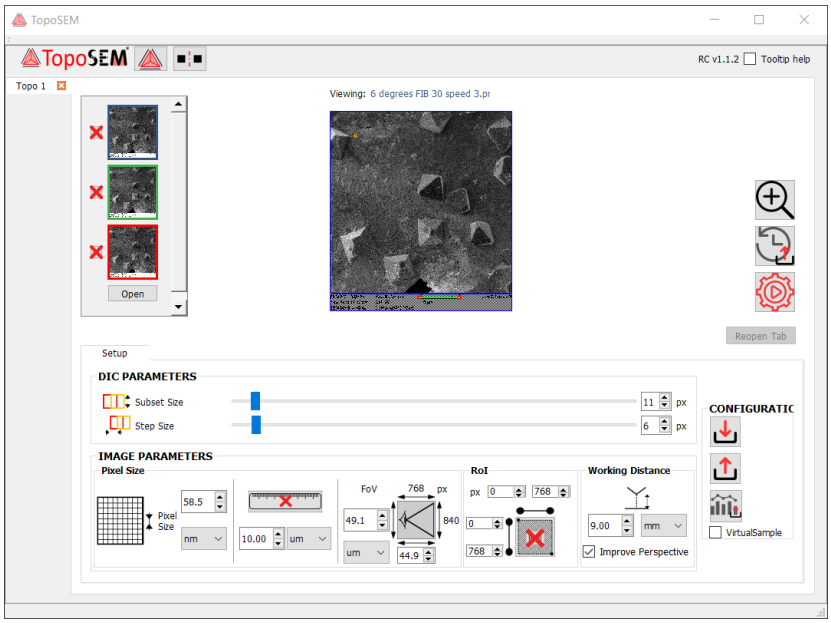


Figure 6: Topographical reconstruction configuration with TopoSEM.

The topographical reconstruction outputs the tilting angles between images and a 3D mesh, based on DIC information calculated between the reference image and the others. The 3D mesh can be exported in a variety of formats to be analysed by other SEM software solutions. Figure 7 shows TopoSEM's output window with an interactive 3D visualiser.

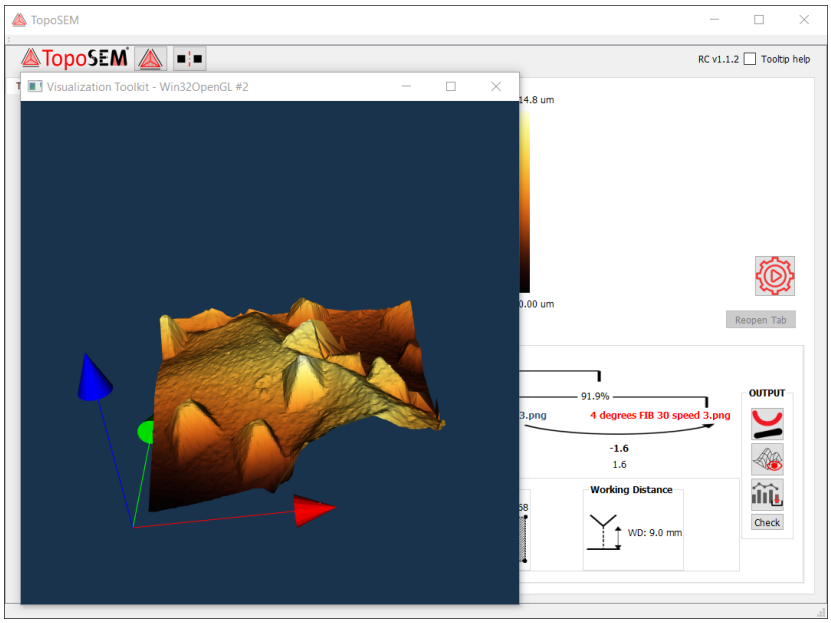


Figure 7: 3D mesh output of the topographical reconstruction with TopoSEM.

3.1.2 *Stability Analysis*

The stability analysis allows the user to input a pair of images taken in sequence of the same sample without tilting the angle to identify non-random noise patterns. Similarly to the 3D reconstruction workflow, the user must also input the parameters related to the image acquisition conditions. TopoSEM then uses these input values, in conjunction with a calculation of the displacements, to obtain numerical analysis of the differences between the images.

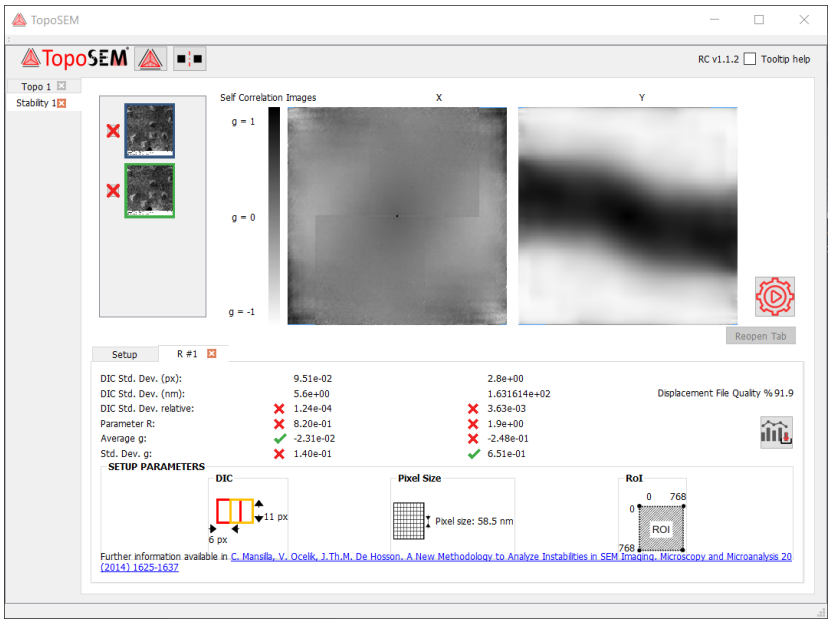


Figure 8: Output of TopoSEM's stability analysis.

Figure 8 shows the output window of TopoSEM's stability analysis. The samples shown were captured by an uncalibrated SEM device and show significant amounts of non-random noise, which has a detrimental impact on the accuracy of a 3D reconstruction. In this case, the device should be calibrated before performing a topographical reconstruction.

3.2 CHALLENGES AND PERFORMANCE OPTIMISATION STRATEGY

The TopoSEM software is designed for two different usage scenarios: in personal laptop/desktop computers, which are usually part of the SEM microscope package with direct access to the captured images; in server environments, where users submit previously collected images to be processed by an online service.

Laptop users only analyse a small amount of images at a time, with the goal of capturing and analysing them quickly. This allows the user to recapture images if the conditions were not adequate for the topographical reconstruction. This is especially relevant since users must request time slots to use the SEM microscopes, which are often small and operationally expensive. In this scenario, these analyses must ideally be performed within a minute to a couple of minutes, thus improving the researcher's productivity. Other tools require much more complex workflows to be performed, which invalidates the possibility of being used when capturing the images. In a server environment, TopoSEM should take advantage of the often large amount of parallel computing resources (cores and/or PUs) to maximise the throughput of processing either larger images, or large amounts of images.

The first step in improving the performance of the current MATLAB implementation is to profile it to get a baseline metric for future improvements. After obtaining the baseline performance the original implementation was ported to C++, a multi-paradigm programming language that includes elements of imperative and object oriented programming and has a great focus on high performance applications. The main advantage of this language when compared with MATLAB is the possibility of directly exploring optimisation features such as vectorization and memory access that are extremely difficult to explore in MATLAB and the high availability of optimised libraries. The port will also allow for a greater modularity and expandability of the code, increasing the ease of introduction of new features. The first step was modifying the existing data structures to reduce the risk of incurring in invalid states. Software bugs often stem from introducing an invalid state. If it is impossible to represent said invalid state it is impossible to introduce it. For example, in the MATLAB version the state of some parameters is represented by a string while it could be represented by a enum. This concept can also be applied to values that are always positive being represented by a unsigned number.

Intel's vTune was then used to identify the limitations of the C++ port (Figure 9). This report shows that the main limitation of the initial port is the core computation of the DIC, representing 99.3% of the total computation time. Inside of the main computation, the IC-GN algorithm is 99.0% of the total computation time. This report also shows that the pre-computation, including image loading and image interpolation, accounts for less than 1% of the total computation time.

Function Stack	CPU Time: Total ▼	CPU Time: Self	Module	Function (Full)	Source File	Start Addre
Total	100.0%	0s				
_start	99.7%	0s	DIC_run	_start		0x402c46
__libc_start_main	99.7%	0s	libc.so.6	__libc_start_…		0x22460
main	99.7%	0s	DIC_run	main	main.cpp	0x402a00
DIC::compute	99.3%	0s	DIC_run	DIC::comput…	dic.cpp	0x404740
DIC::analyse_neighbour	99.0%	0s	DIC_run	DIC::analyse…	dic.cpp	0x404500
ICGN	99.0%	0s	DIC_run	ICGN(Config…	icgn.cpp	0x4070a0
get_delta_p	96.0%	2.956s	DIC_run	get_delta_p(…	icgn.cpp	0x406650
GridPoint::get_hessian	2.9%	0.140s	DIC_run	GridPoint::ge…	gridpoint.cpp	0x405a50
get_updated_p_vector	0.1%	0s	DIC_run	get_updated…	icgn.cpp	0x406e20
is_out_of_bounds	0.0%	0.120s	DIC_run	is_out_of_bo…	icgn.cpp	0x407152
[Unknown stack frame(s)]	0.0%	0s		[Unknown st…		0
GridPoints::find	0.0%	0s	DIC_run	GridPoints::fi…	gridpoints.…	0x405f50
ICGN	0.3%	0s	DIC_run	ICGN(Config…	icgn.cpp	0x4070a0
DIC::DIC	0.5%	0s	DIC_run	DIC::DIC(std:…	dic.cpp	0x4034f0
__clone	0.2%	0s	libc.so.6	__clone		0xfe990
_dl_start_user	0.0%	0s	ld-linux…	_dl_start_user		0x1148

Figure 9: Top-down tree graph generated by vTune's hotspot report for the initial C++ port.

Data structure

The first step of the computation of the full field displacements is loading the reference and target image provided by the user. This step is performed with the usage of the OpenCV library where the built-in functions allow the loading of a given image into a Matrix object. Internally this object contains a vector with all the values from the matrix organised line-by-line. Then a bi-cubic interpolation is applied to the reference image and the target image with the aim of achieving sub pixel precision. This step creates a image in memory with higher resolution than the original.

Based on the configuration and the loaded images the DIC library pre-computes a vector of grid points. Each grid point is represented by the centre point, the minimum and maximum row and column, the current status of the point (i.e. if it has already been computed), and the the computed displacement. It is possible to, given the coordinates of the centre point, obtain a reference to the grid point from the vector. This was done in order to minimise the overhead of creating grid points during the main computation process. The data structure that stores a single grid point occupies 68 bytes.

compute

The main function where the DIC algorithm is computed is named $\mathrm{compute}$ (Figure 11). By analysing the output of vTune it is confirmed that this is the function where the majority of the computation resources are used. In the initial port the total of time spent is 99.5% of the total time.

The Figure 10 represents the order in which the full field displacment is computed. In this image each square represents a grid point and each circle inside each square represent the centre point of that corresponding grid point. The circle in red is the initial guess provided to the library and the circles in blue represent the neighbours of the initial guess in the directions: Up, Down, Left and Right.

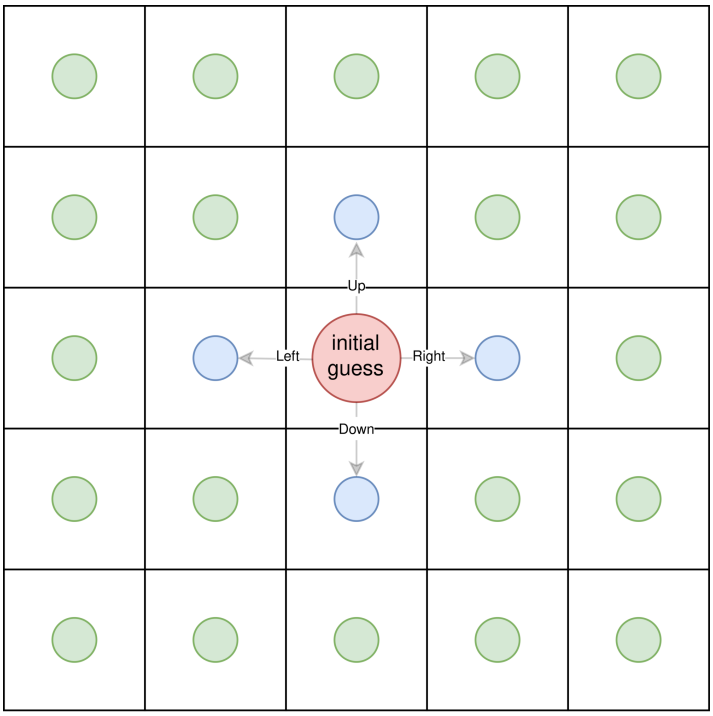


Figure 10: Order of sequential computation of the full field displacement.

Figure 11 presents the workflow of the $\mathrm{compute}$ function. Initially, it receives a initial guess (circle in red) as a parameter in the form of a grid point and pushes it to a queue. Then a loop is started where the program checks if the queue is empty. If it is empty the computation has finished. If it is not empty it removes a element from the queue, then it computes the immediate neighbours of said grid point (in the case of the initial guess the circles in blue).

To do this the function first attempts to find that neighbour in the pre-computed vector of grid points. If it can not find the neighbour in the vector, meaning that it is outside of the image (out-of-bounds), or if that neighbour has already been computed it is skipped. To compute a neighbour the value of the displacement for the removed point is used as initial guess. The $ICGN$ function (Figure 12) is then called on the neighbour grid point using the displacement of the central point as initial guess. The computed grid point is then pushed to the queue and the loop restarts by trying to remove another element from the queue and processing its neighbours until the queue is empty.

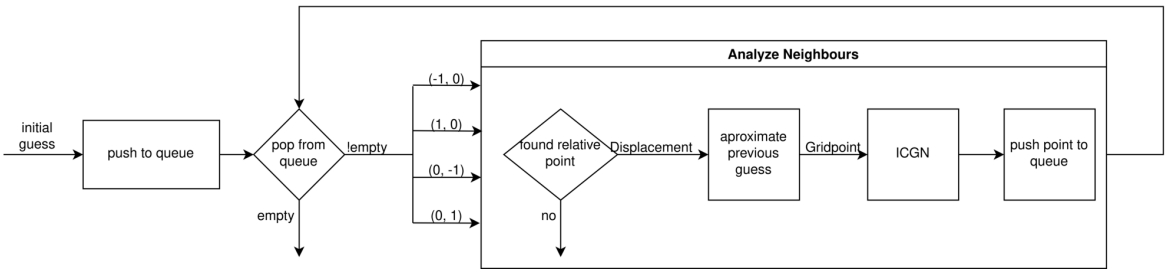


Figure 11: Execution flow diagram of the sequential version of the main compute function.

IC-GN

The $ICGN$ function (Figure 12) is where the iterative process of refining the initial guess provided for finding the maximum of the correlation array described in Section 2.1 is computed. This function accounts for 99.1% of the computational time of the initial version of the C++ port using the case study presented in Section 4.1.

The function first computes the hessian matrix for the given grid point via the $\mathrm{get\_hessian}$ function, represented by the execution flow diagram in Figure 14. Then the iterative process is started. If the current displacement guess results in a out-of-bounds subset in the target image or if any value in the displacement is Not-a-Number (NaN) then the function exits with a error. Then a new displacement guess using the current one, the Hessian and the current subset are computed with the help of the $\mathrm{get\_delta\_p}$ function, which is presented in Figure 13. If the result has converged then the iterative portion has ended. The displacement is then used to obtain the corresponding pixels on the target image. To quantify the difference between the reference subset and the target subset the $ZNSSD$ algorithm (Equation (3)) is then used.

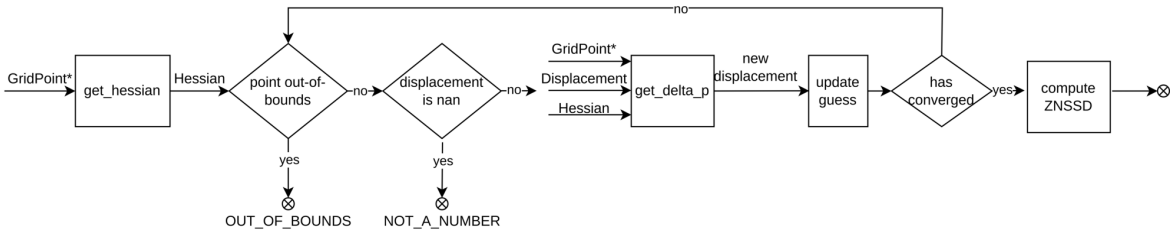


Figure 12: Execution flow diagram of the $ICGN$ function.

get_delta_p

The Figure 13 represents the execution flow of the $\mathrm{get\_delta\_p}$ function. It is called once for each iteration performed for each grid point in the computation and in total accounts for 96.1% of the computation time of the initial port being the main culprit for the subpar computation time. It consists of iterating over all the values in: the subset in the original image; the subset on the target image; the subset in the original image intensity; and the subset in the target image intensity. In this process a new matrix is computed that is then multiplied with the negation of the inverse of the Hessian matrix passed as argument. Since the $\mathrm{get\_delta\_p}$ function consists of matrix operations it is a prime target for vectorization. These matrix operations were implemented in the initial port by using OpenCV. However, trough the analysis of the compilation report from the compiler it was possible to identify that none of the loops for iterating over the subsets were vectorizing. In fact, OpenCV masks the fact that the data structures are internally contiguous from the compiler, inhibiting any and all vectorization.

The OpenCV library supports dereferencing entire lines of data instead of dereferencing for each individual access. By changing the code to perform the dereference only once per line it is possible to vectorize the computation of each line. Furthermore, by changing the loop order and transposing the matrices by changing the access order it was possible to explore cache locality in the main loop of the $\mathrm{get\_delta\_p}$ function. After performing these optimisations, the $\mathrm{get\_delta\_p}$ function now represents 0.4% of the overall computation time.

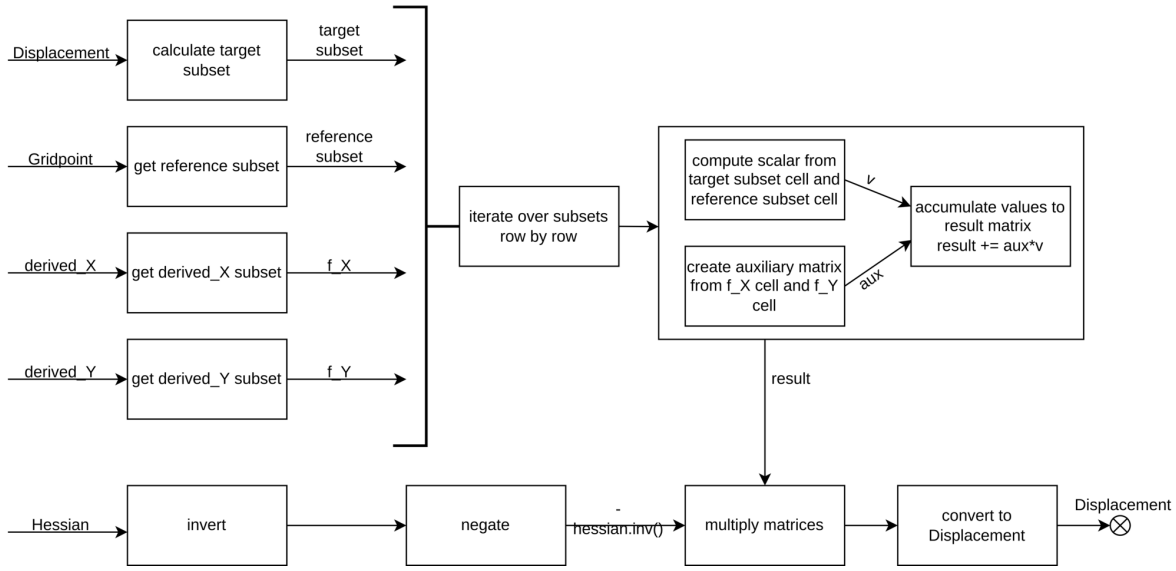


Figure 13: Execution flow diagram of the get_delta_p function.

get_hessian

The get_hessian function is called once for each grid point in the computation. By using the vTune software to again identify the main limitations of the new version after the get_delta_p optimisation it was discovered that it represents 96.8% of the total computation time. Before the last optimisation step this function represented less than 2% of the computation time. Figure 14 represents the execution flow of the get_hessian function. This function consists of iterating over the values of the the subset in the original intensity image; and the subset in the target intensity image to accumulate the values into the hessian matrix. The process of accumulation involves transposing and multiplying matrices and was initially implemented by using the OpenCV built-in functions for matrix operations. Similar to the issues of the get_delta_p function, the usage of OpenCV was the reason why the iteration over the subsets did not vectorize. By eliminating the usage of OpenCV inside the main loop it was possible to reduce the total computation time of the function to 2.0%.

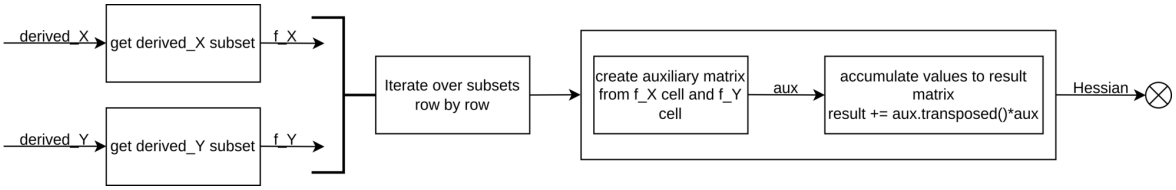


Figure 14: Execution flow diagram of the get_hessian function.

find grid point

As previously described, a vector of grid points is the main data structure of the DIC computation. To find the corresponding grid point based on it's centre coordinates a find function was created. The first implementation of this function consisted of iterating over all the elements of the vector of grid points until it found the corresponding

centre coordinate. This means that the initial implementation had a complexity of $O(n)$ where n is the number of grid points. Initially the find function had no significant impact on the computation time. However, after optimising the functions described above it represents 8.6% of the total computation time, making it the function with the highest percentage of the total computation time that has yet to be optimised.

When the grid point vector is generated the grid points are inserted line by line. This means that the spacing between them is regular and defined by the configuration made by the user of the library. Therefore, it is possible to generate the index of a given grid point based only on the configuration and the coordinates of the centre of the grid point. By using Equation (6) it is possible to compute the index of the grid point in the vector, making the new implementation $O(1)$. The find function became residual and does no longer appear on the vTune hotspot report.

$$\frac{x - spacing\_top}{grid\_spacing.height()} * n\_horizontal + \frac{y - spacing\_left}{grid\_spacing.width()} \tag{6}$$

where:

- (x, y) are the centre coordinates of the grid point;

grid point vector creation

After performing the aforementioned optimisations and analysing the vTune report for hotspots for the new version it was discovered that the creation of the grid point vector represents 21.1% of the total computation time.

The size needed for the grid point vector is equal to the total number of grid points and can be computed before it is created since it is based on the resolution of the images and the configuration provided by the user. Thus, it is possible to pre-reserve the space needed in the vector before inserting the grid points.

A new version was created that makes use of the $\mathrm{std::vector::reserve}$ to change the vector capacity to the total number of grid points. This was done to mitigate the impact that comes from the memory reallocation when inserting elements into a dynamic data structure and resulted in the computation time of the creation of the grid point vector to reduce to 17.8%. However, this improvement was offset by the cost of pre-allocating the memory resulting in an overall negligible performance improvement

Workload Parallelization and Scheduling

After sequentially optimising the code and vectorizing the main hotspots of the DIC library, the focus shifted towards exploring the performance available in modern multicore homogeneous computing environments.

To be able to compute the displacement for a given grid point there are only two values that are needed: **(i)**the grid point that will be computed and **(ii)**the initial guess that is based on the closest neighbour that has already been computed. Since it is possible to obtain the corresponding grid point based solely on its coordinates, it is possible to use the coordinates of the current grid point and of the parent grid point to obtain these values. Therefore the minimum amount of information a worker needs to know which grid point it needs to compute are a pair of coordinates for the current and parent grid point.

The Figure 15 represents the order of computation and data dependency between the grid points of an image. The computation of the full field displacement starts from a central grid point (circle in red) passed as arguments known as initial guess . Then, the neighbours of the initial guess (circles in blue) can be computed because there is a neighbour whose displacement has already been computed. This dependency in the order of computation is represented by the arrows in grey. Only after this first wave is computed can the second wave (circles in green) be computed. The dependency continues for all subsequent waves meaning that the computation of a given wave is limited by the computation of all the previous waves. This means that grid points outside of each dashed line can be computed after all the grid points inside that same line have already been computed. This makes the dashed line a barrier for the computation. The data dependency described means that the computation propagates from the initial guess outwards and this kind of dependency is often called Front Wave Propagation.

In the Figure 15 each circle represents the centre of the corresponding grid point and contains a letter and a number. The letter represents the order in which each wave will be computed. The number represents the order in which the grid points will be computed inside each wave. As already stated above the grid points are stored in a vector line by line. By computing the wave by following the order of line by line it allows the program to explore the data locality in the cache when accessing the data structure.

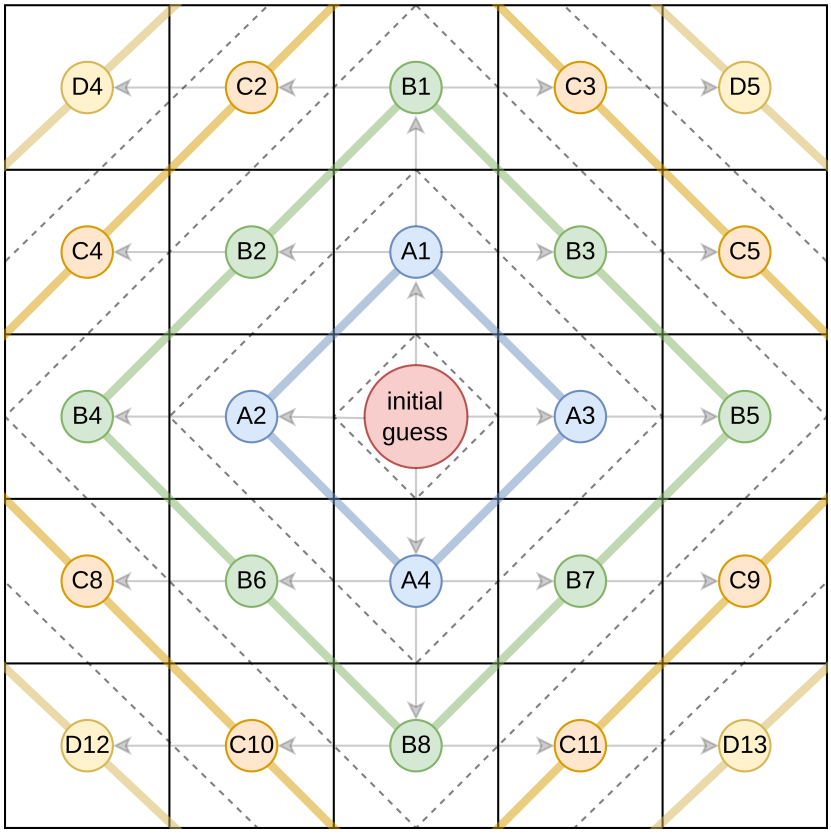


Figure 15: Order of parallel computation of the full field displacement.

This behaviour cannot be represented by the scheduler types available on OpenMP and further optimisation steps would be limited by the architecture of OpenMP. Images generated by current SEM technology have a limited resolution and even accounting for future developments in the field of SEM imagery, the performance penalty associated with OpenMPI's data communication far outweighs its performance improvements. The ap-

proach chosen for the first step in parallelization was the creation of a scheduler with a variable number of worker threads using pthreads that implements the aforementioned propagation behaviour. Since there was the possibility that the datatype used to represent a segment of work could change, the scheduler was designed to be generic and standalone.

The main computation thread is used to manage work distribution and to generate new work. The function responsible for this is called $\mathrm{generator\_daemon}$, represented by the execution flow in the Figure 16. It receives a function as argument that is able to compute a given wave based solely on its identifier. Then a loop is started where the generator calls the function for wave generation. If it returns a wave, the wave is pushed to an internal locked work queue and all workers are notified that there is new work available. The available queues in the C++ standard library do not have support for multi-thread and other libraries do not support the feature set needed for this environment. Thus, a novel implementation of a locked queue that supports one main thread inserting waves (i.e. vectors of work) and many threads requesting work was created.

Every time a worker finishes computing an assigned batch of work it notifies the scheduler. The scheduler is waiting after pushing the wave to the work queue for said signals from the worker threads. If the current wave computation was finished the scheduler generates a new wave and notifies the workers. The cycle continues until the wave generation function returns None. In this case, the scheduler is closed and all workers are notified.

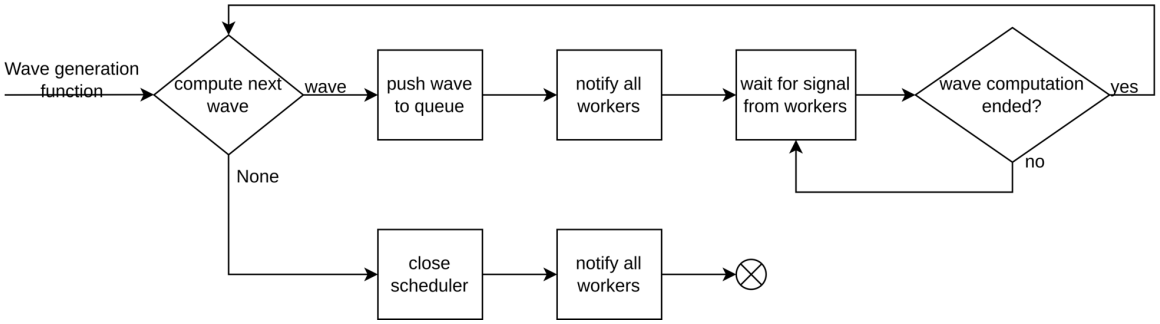


Figure 16: Execution flow diagram of the $\mathrm{generator\_daemon}$.

Before starting the $\mathrm{generator\_daemon}$ the DIC library spins up N worker threads. Each worker thread, represented by the diagram in the Figure 17), attempts to get work from the queue in the scheduler. The work request is represented by the execution flow diagram in the Figure 18. If it is able to get work it finds the corresponding current grid point and parent grid point. Then it approximates the new guess based on the displacement of the parent grid point. This displacement is then used in conjugation with the current grid point as arguments for the $ICGN$ function. After finishing the computation of the fetched work the worker thread notifies the scheduler that it finished the work and attempts to fetch more work. If the scheduler is closed it means the computation has ended and the worker thread exits.

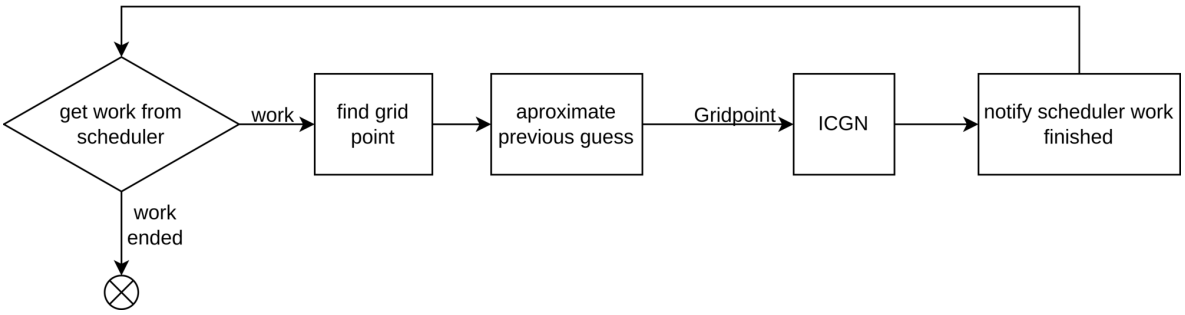


Figure 17: Execution flow diagram of the worker thread with the initial scheduler.

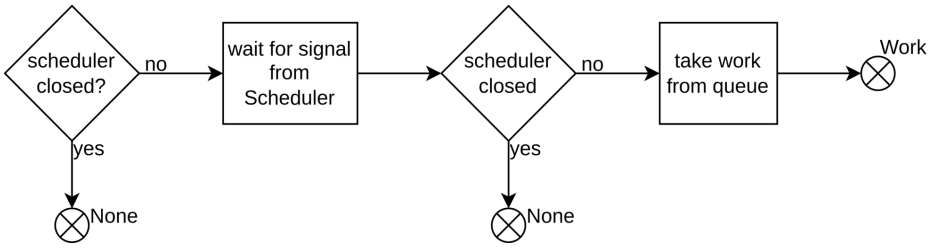


Figure 18: Execution flow diagram on getting work from the initial scheduler.

The distribution of work between five threads is represented by the diagram of the Figure 19 The scheduler starts by generating a new wave and then the threads request work from the scheduler. However, the first wave only contains four elements of work and thus one of the five threads is starved. This behaviour is represented by the circle with a cross inside. After, the scheduler thread generates new work and the cycle is repeated until all waves have been processed.

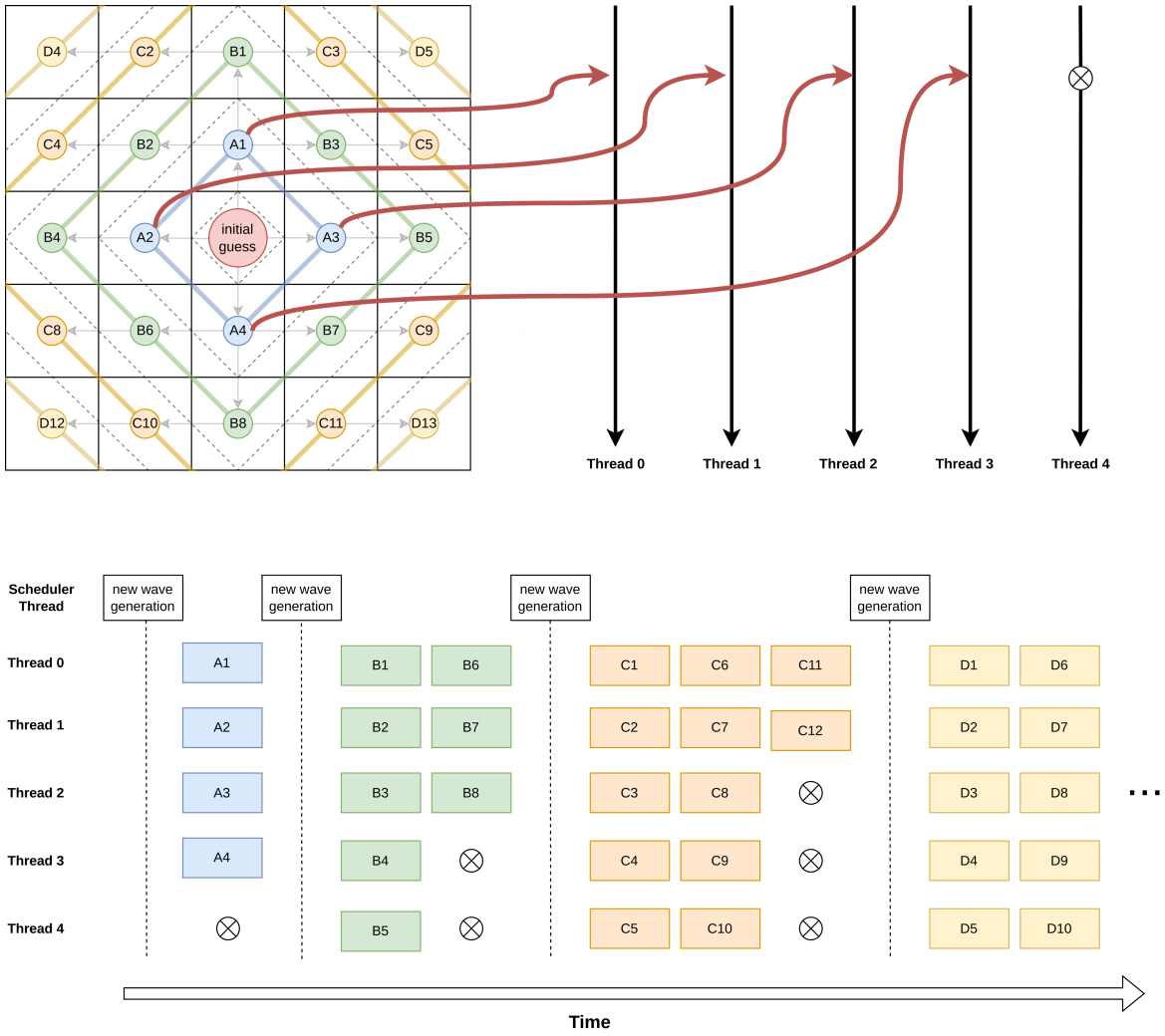


Figure 19: Scheduling of grid points with five worker threads.

This scheduler introduces an overhead in the computation when a worker thread is requesting work. If the main thread is currently generating new work or if another worker thread is requesting work from the scheduler, the thread currently requesting more work will need to wait. It is also possible for more than one thread to be starved when the the number of work in the current wave being computed is not divisible by the number of threads.

Batch Scheduler

With the aim of reducing the aforementioned overhead associated with requesting work, a new version of the generic front wave propagation scheduler developed on the previous step was created. When a worker requests work from the scheduler, instead of receiving a single piece of work, it can receive a batch containing a predetermined amount of pieces of work all from the same wave to guarantee that there is no data dependency between them. This size is static and is defined prior to the execution of the scheduler. Now the worker threads need to request work less times from the scheduler leading to a reduction of the total overhead.

To achieve this behaviour the generator_daemon only needs minor modifications. Namely, a worker thread now notifies the scheduler that it finished an entire batch. Yet, the worker threads need more extensive modifications to support static batching. The behaviour of the new worker threads is described by the diagram in the Figure 20. Instead of computing a single piece of work the worker iterates over the batch to compute all the segments of work contained inside.

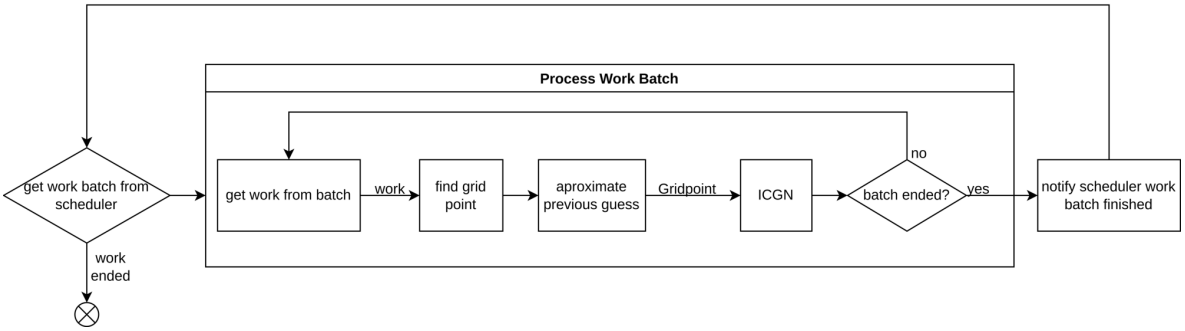


Figure 20: Execution flow diagram of the worker thread with the batch scheduler.

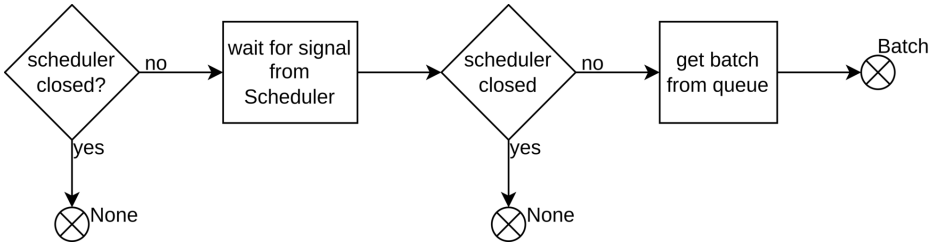


Figure 21: Execution flow diagram for getting a batch from the batch scheduler.

This new version allows for parameter fine tuning. This means that depending on the computing environment it is possible to choose the batch size and worker thread count that maximises performance. The feasibility of this optimisation step will be explored in the next chapter (Chapter 4).

The Figure 22 represents the behaviour for the scheduler with batch size of four with five worker threads. The scheduler starts by generating a new wave. Then the threads request a batch of work with size four from the scheduler. However, the first wave only contains four elements of work and thus only one thread is used for the computation of the first wave. After the first wave is computed, the scheduler thread generates new work and the cycle is repeated until all waves have been processed.

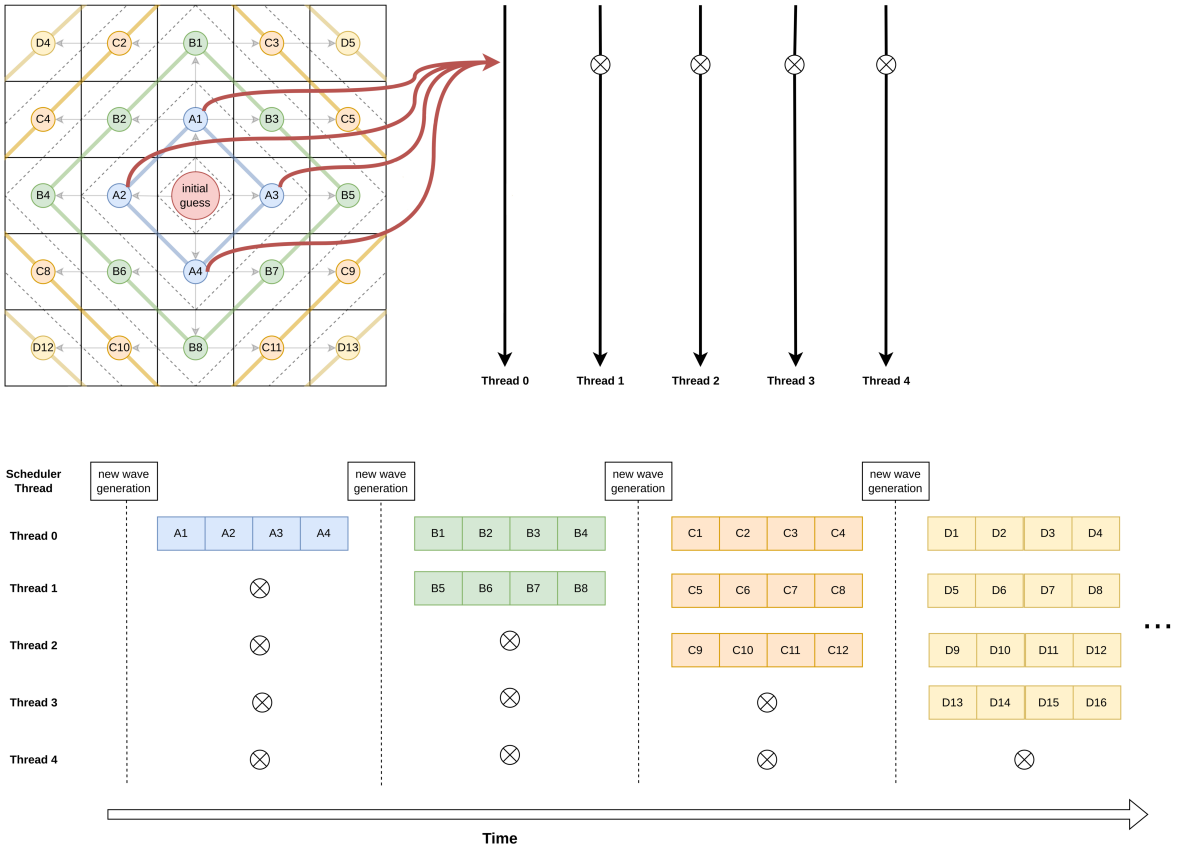


Figure 22: Relation of the batching scheduler (with batch size of four) with five worker threads

As it is possible to see from the Figure 22, for the initial waves there is a greater amount of thread starving when compared to the simple scheduler. However, the overhead of requesting more work is reduced when compared to the simple scheduler. In the Chapter 4 it will be analysed if the performance improvement from reducing the requesting overhead outweighs the negative impact from increased thread starving.

3.3 FUNCTIONAL VALIDATION

With the goal of validating the DIC port to C++ a set of unit-tests were created. Unit-testing is a method of testing software that focuses on writing tests for small sections of the program (usually single functions or classes). Thus, each test is smaller and easier to write than other testing methodologies.

The main candidates for a unit-testing framework were Catch, Boost.Test and Google Test. According to Jetbrains (2021), Google Test is, by far, the most used unit-testing framework in C++ with 32% of programmers using it, followed by Catch with 11% and lastly Boost.Test with 5%. The latter is also the most complex to use and thus was discarded as a candidate. Google Test and Catch have a similar set of functionalities. However, Google Test revealed to be easier to include and use, making it the unit-testing framework used in this dissertation.

The unit-tests created compare the results produced by the main functions of the port to the results produced by the original MATLAB implementation of the same function.

4

RESULT ANALYSIS

4.1 TESTBED AND METHODOLOGY

As already stated before, one of the goals of this library is to be able to run efficiently on a great variety of computing environments with distinct computation power and specifications. Therefore, in order to present meaningful results, a set of different test beds were prepared to represent some real-world scenarios this library will face.

To represent personal laptops often available on a Scanning Electron Microscope (SEM) laboratory, a machine with an Intel i7-7700HQ (4 cores, Kaby Lake, 2.8 GHz, 16 GiB RAM), Linux 5.15.6-arch2-1, and OpenCV 4.5.5 was be used. This machine will be referred in this chapter as *laptop*.

To represent a server/cluster, two distinct computing nodes were selected from the *Services and Advanced Research Computing with HTC/HPC clusters* (SeARCH[1]) at the University of Minho.

With the aim of analysing the impact of dual socket architectures, a node with higher core count was chosen. The node has a dual socket Intel Xeon E5-2695 (12 cores, Ivy Bridge, 2.4 GHz, 64 GiB RAM), Linux 3.10.0-1160.41.1.1l7.x86_64 and OpenCV 4.6.0. This node will be referred to as *C24*.

In order to further studying the impact of faster single core performance, a node with lower core count but higher base clock was chosen. The node has a dual socket Intel Xeon E5-2650v2 (Ivy Bridge, 8 cores, 2.6 GHz, 64 GiB RAM), Linux 3.10.0-1160.41.1.1l7.x86_64 and OpenCV 4.6.0. This node will be referred to as *C16*.

Two different pairs of SEM images were chosen to benchmark the program during its development. A pair of images with a higher resolution of 2576 pixels by 2086 pixels (Figure 23), referred to as large dataset, and a pair of images with smaller resolution of 768 pixels by 840 pixels (Figure 24), addressed as small dataset. The initial guesses used for both pair of images were determined by using the GUI of the original MATLAB implementation and are the same across all the benchmarks.

1 https://www4.di.uminho.pt/search/pt/index.htm

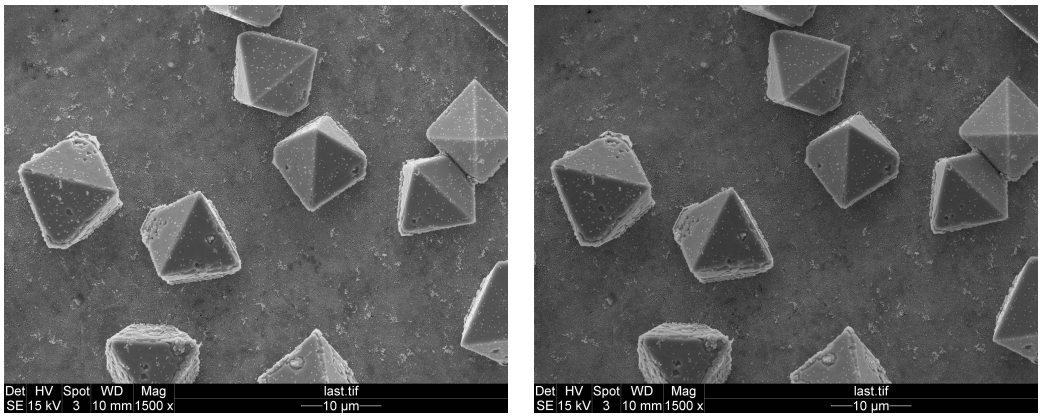


Figure 23: Pair of 2576x2086 images (Large dataset) used for DIC benchmarking.

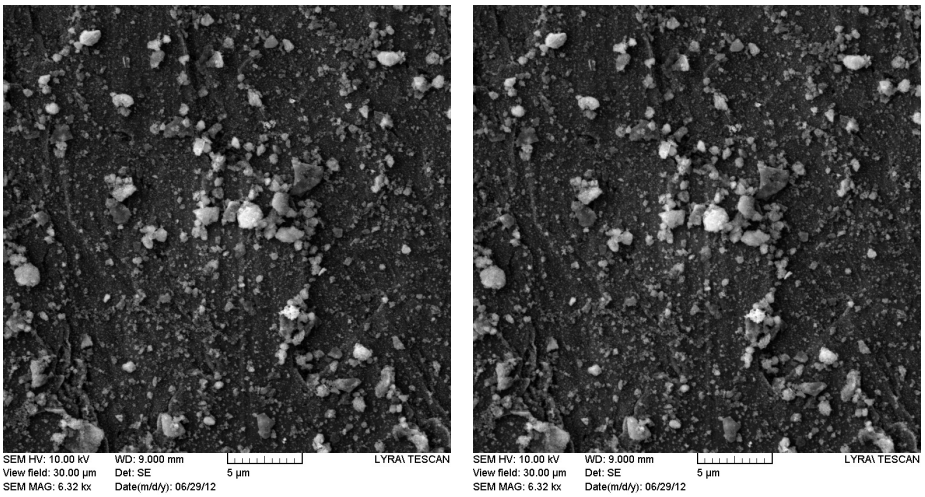


Figure 24: Pair of 768x840 images (Small dataset) used for DIC benchmarking.

The measurements presented on this chapter were performed using the K-Best method, described in Patterson and Hennessy (2013). Even tough external factors might negatively impact the execution time of a given program, it never executes faster than the theoretical capabilities of the hardware. Hence, when benchmarking a computer program, if the average of the measured values is used the result will not be representative of the real world performance of the algorithm. This happens because there are no values that are better to cancel out the worse results. This leads to the average of all values being a worse value than in reality. The K-Best method aims to mitigate this issue by using the following steps:

1. measure the computation time N_min times;
2. get the K smallest measurements;
3. if the difference between the best and the k-nth best is less than ERROR% got to last step, else go to next step;
4. measure computation time one more time;

5. if total number of measurements is superior to N_max the program is unstable and the measurements are invalid, else return to step 2;

6. compute the average of the K smallest measurements;

By only using the average of the K best values that are inside a predetermined interval given by the ERROR, it is possible to eliminate the runs affected by external values and statistical outliers. Thus, it is possible to obtain results closer to the real world performance. In this chapter the values used for K-Best are: K = 3, ERROR = 5%, K_MIN = 10, K_MAX = 20.

4.2 PERFORMANCE ASSESMENT

The Table 1 shows the benchmarks that were executed for the MATLAB implementation on the laptop. This implementation takes over 54 minutes to compute the large dataset and takes over 7 minutes to compute the small dataset. Since the need to improve the performance of the MATLAB implementation is already clearly highlighted by this benchmarks and MATLAB was not available on the SeARCH cluster, the code was not executed on the C16 and on the C24.

	laptop	C16	C24
small dataset	**449,11**	-	-
large dataset	**3279,5**	-	-

Table 1: Execution time (in seconds) for the MATLAB implementation.

Initial Port

The performance of the initial port greatly improves on the performance of the original MATLAB implementation, achieving a speedup of 38,03 for the small dataset and 31,23 for the large dataset. Table 2 presents all the benchmarks that were performed using the K-Best method on the laptop, C16 and C24 using both the large dataset and the small dataset as input. Since the sequential implementation uses only one core to execute it is dependent on the single core performance of the PU being used. In this case, the PU with the fastest base clock speed is the *laptop* followed by C16 and finally C24. The turbo boost available on the laptop's PU is also more aggressive and results in a significant improvement over the base clock when it triggers for short time frames. Furthermore, the architecture in the *laptop*'s PU is more recent (Kaby Lake was launched on 2016 vs Ivy Bridge was launched on 2012) and includes optimisations for better performance, namely for operations over vectors. The difference in architecture, base clock speed and boost clock speed translates in the *laptop* being by far the fastest in executing the initial port, followed by the C16 and C24, in this order.

	laptop	C16	C24
small dataset	**11,81**	19,50	20.93
large dataset	**105,01**	172.59	181.63

Table 2: Execution time (in seconds) for the initial C++ port.

get_delta_p optimisation

The first optimisation step was improving the get_delta_p function. The Table 3 compares the computation time of the new version with the initial port and gives the speedup achieved. This optimisation lead to a speedup of up to 1.37 for the small data set and 1.29 for the large image dataset when compared to the initial C++ port. The trend of the fastest machine being the *laptop* continues in this optimised version. For the small dataset executing in the *laptop* the speedup was 0.99 which is within the margin of error of the measurements and can be considered that there was no speedup in this specific circumstances.

		laptop	C16	C24
small dataset	original	11,81s	19,50s	20,93s
	optimised	11,85s	14,49s	15,31s
	speedup	**0,99**	**1,35**	**1,37**
large dataset	original	105,01s	172,59s	181,63s
	optimised	76,61s	143,74s	140,94s
	speedup	**1,37**	**1,20**	**1,28**

Table 3: Execution time (in seconds) and speedup for the get_delta_p optimisation.

The standard deviation is a metric that is able to express by how much the members of a group differ from the mean value for the group and is given by the Equation (7). The lower the standard deviation of the runs of a benchmark the closer the results of that benchmark were. The standard deviation for the initial port on the *laptop* is much larger than for the get_delta_p optimisation (Table 4). Seeing that one of the requirements for this library is that it can execute efficiently both on laptops and clusters, even tough this optimisation might not represent a speedup on all situations it will represent a better user experience and less variation on the computation times. There are two possible explanations for the fact that both C16 and C24 have a lower standard deviation than the laptop. Firstly, personal laptops have background tasks that can affect individual benchmark runs whereas servers usually only have one task running. Secondly, as already stated above, modern laptops have Turbo boost which can boost the performance of the PU on short burst but it is not guaranteed to startup on every single run. Acun et al. (2016) analysed the impact of Turbo Boost on the stability of the results of benchmarks performed on machines with and without Turbo Boost and found that, whereas machines without turbo boost had only 1% variation, machines with turbo boost had up to 16% variation.

		laptop	C16	C24
small dataset	original	**67,53**	4,32	6,23
	optimised	**18,21**	3,25	7,81
large dataset	original	**29,99**	0,25	0,62
	optimised	**1,49**	0,59	2,64

Table 4: Standard deviation for the get_delta_p optimisation.

$$s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{7}$$

where:

- s population standard deviation
- N the size of the population
- x_i each value from the population
- $\overline{x}$ mean of the population

get_hessian optimisation

After optimising the $\mathrm{get\_delta\_p}$ function, the $\mathrm{get\_hessian}$ function became the function that represented the largest percentage of the total computation time. The optimisations previously described resulted in a a speedup of up to 24.38 for the small data set and 12.92 for the large data set when compared to the initial port.

		laptop	C16	C24
small dataset	original	11,81s	19,50s	20,93s
	optimised	0,57s	0,80s	0,87s
	speedup	**20,72**	**24.38**	**24.06**
large dataset	original	105,01s	172,59s	181,63s
	optimised	8,13s	13,53s	14,90s
	speedup	**12.92**	**12.76**	**12.19**

Table 5: Execution time (in seconds) and speedup for the $\mathrm{get\_hessian}$ optimisation.

find optimisation

This optimisation step has a greater impact on the large dataset than on the small dataset (up to 31.44 for the large dataset when compared to up to 27.46 for the small dataset) (Table 6).

Before optimising the computation cost of the find function was linear and proportional to the number of subsets. After optimising, the find function is constant and independent from the number of subsets. Further, this function is called as many times as there are subsets for a given DIC computation. Hence, in the initial find version, the larger the image, the higher the computation time of the find function and the more times it was called. Meaning that in the new version, the larger the image the more total time is saved.

		laptop	C16	C24
small dataset	original	11,81s	19,50s	20,93s
	optimised	0,52s	0,71s	0,77s
	speedup	**22.71**	**27.46**	**27.19**
large dataset	original	105,01s	172,59s	181,63s
	optimised	3,34s	5,84s	6,39s
	speedup	**31.44**	**29.55**	**28.42**

Table 6: Execution time (in seconds) and speedup for the find optimisation.

Parallelization and Workload Scheduling

Although the execution time tends to be lower on the *laptop* for single threaded execution, this trend is reversed for the parallel implementation of the DIC algorithm. The overhead of implementing the batching scheduler is minimal. Therefore there is no difference between analysing the performance of the first implementation of the scheduler and the batching implementation of the scheduler with batch size of one. Thus, the performance analysis of the parallel implementation will focus on the final implementation with the batch scheduler.

Contrary to the initial expectations, the introduction of a batch scheduler tends to decrease the performance of the algorithm when the size of the batch is greater than one in every machine when the worker thread count is greater than one for every data set. Increasing the batch size creates larger work chunks and reduces the granularity of the work distribution. The only situation where there is some benefit across the benchmarks from increasing the batch size is when there is only one worker thread. Since there is no need for greater granularity on the work division there is no negative impact from increasing the batch size. Further, the batch size increase leads to less requests from the single worker thread reducing the overhead of the scheduler and improving performance. However, this case is not relevant as an optimisation strategy because increasing the thread count from 1 to 2 improves the computation time far more than increasing the batch size. This behaviour is specially noticeable for the small data sets. These benchmarks are represented by the graphs in the Figure 25 and Figure 26 where the x-axis is an increase in worker thread count, the y-axis represents the execution time measured using the K-best method and the different plots represent different maximum batch sizes.

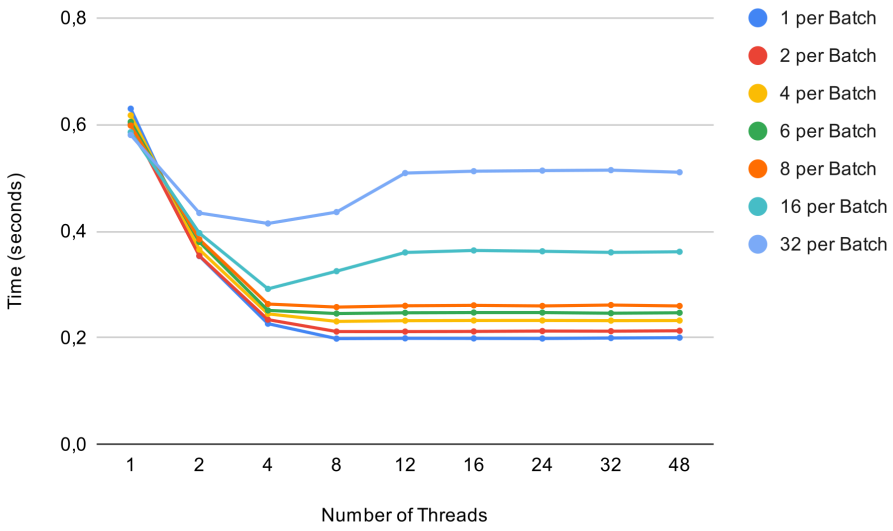


Figure 25: Execution time (in seconds) for the batch scheduler on laptop with varying thread count and batch size for the small dataset.

With the use of the scheduler the computation load is divided across each worker thread. Increasing the worker thread count leads to a reduction of the computation load of each worker thread. However, there is a cost associated with spinning up each worker thread that offsets the benefit that comes from parallelisation. If the computation load of the worker is not enough to amortise the spin-up cost there is a reduction in performance. In extreme cases, this leads to a curve, known as bathtub curve, where increasing the count of the worker threads

actually results in increasing the computation time instead of decreasing it. This behaviour can be seen on the graph on the right in the Figure 26, that represents the benchmark executed on the C24 for the small dataset, when there are only 1 or 2 elements per batch starting at 16 worker threads. It is also visible starting at 4 worker threads for 16 and 32 elements per batch on the graph Figure 25 representing the benchmark on the *laptop* for the small dataset.

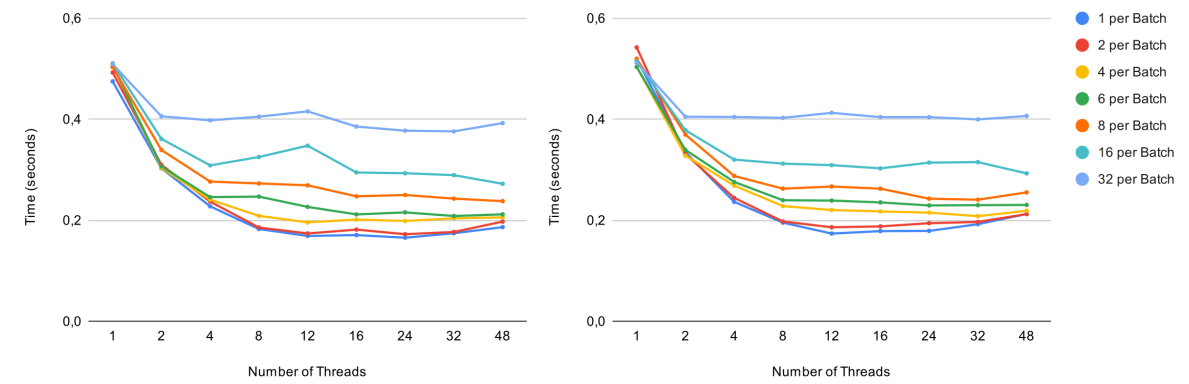


Figure 26: Execution time (in seconds) for the batch scheduler on C16 (left) and C24 (right) with varying thread count and batch size for the small dataset.

An increase in worker thread count does not have the same impact on the small dataset as on the large dataset, the impact on the larger dataset is greater. If the image has a higher resolution there will be more subsets, more waves and larger overall waves. This means that it is possible to distribute more work across more threads while each thread has enough assigned work to amortise the cost of being created. The aforementioned behaviour is visible when comparing the greater improvement that comes from increasing the number of worker threads from 8 to 12 on the C16 for the large dataset when comparing to the small dataset. The graph on the left of Figure 26 represents the computation time for the small dataset on the C16 and the graph on the left of Figure 28 represents the computation time for the large dataset on C16.

Similarly, the impact of increasing the worker thread count is different on different computing environments. If the dataset is the same and the number of cores on the computing environment increases, increasing the worker thread count can yield a better improvement on the machine with the larger core count. This occurs because resource starving can happen if the worker thread count is larger than the core count of the computing environment. For example, when comparing the benchmarks on the Figure 28 executed on the C16 and C24 with the benchmarks on the Figure 27 executed on the *laptop*, where a reduction of core count occurs, the result of increasing the worker thread count from 8 to 12 also leads to less performance improvement. This happens because the worker thread count after the increase is greater than the number of threads on the *laptop* and the same does not happen for the C16 and C24. Therefore, it is possible to conclude that the bathtub curve behaviour is exacerbated by smaller workloads and reduced core count.

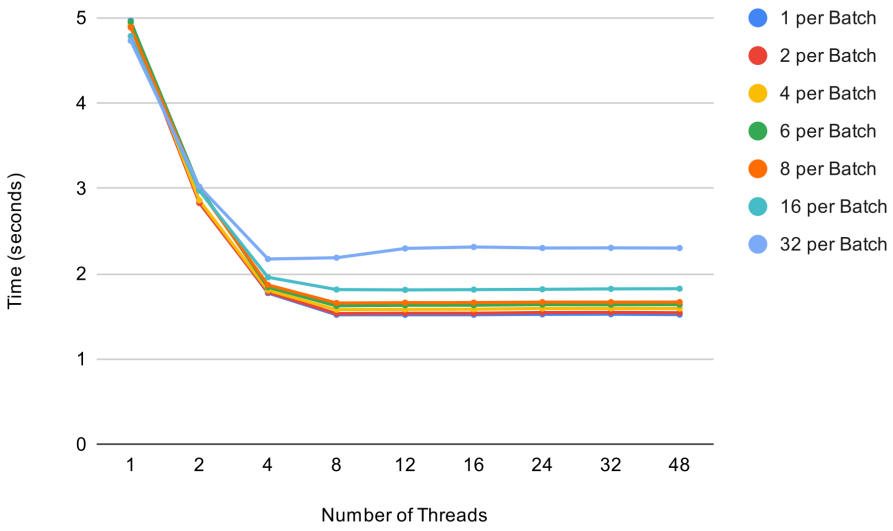


Figure 27: Execution time (in seconds) for the batch scheduler on laptop with varying thread count and batch size for the large dataset.

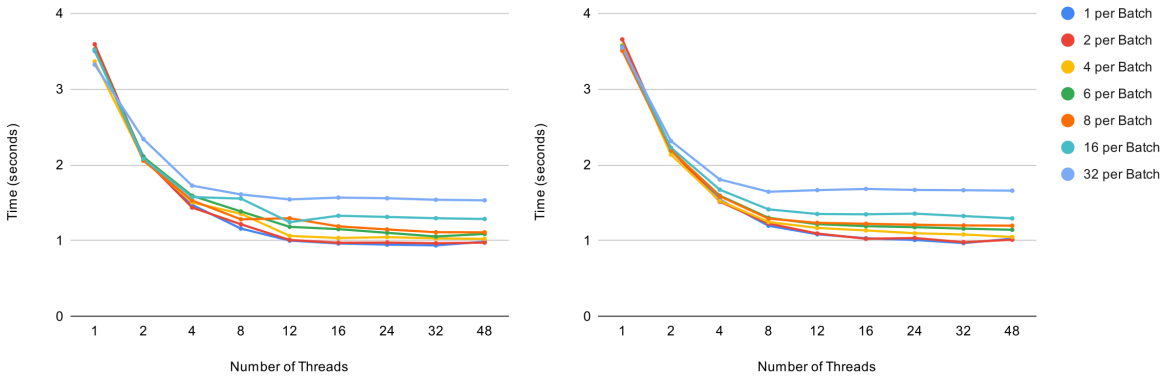


Figure 28: Execution time (in seconds) for the batch scheduler on C16 (left) and C24 (right) with varying thread count and batch size for the large dataset.

5

CONCLUSION

Digital Image Correlation is a technique capable of analysing the deformations between two images along a set of predetermined points. The deformation data is then used by the TopoSEM to produce both a stability report of the SEM's calibration and to create a 3D reconstruction of a sample.

A novel DIC implementation was developed by Electron Softview, in MATLAB to be used in the TopoSEM software package. However, the performance of this implementation is subpar which is a problem since there is a limited time frame for a researcher to use a SEM. Thus, the main focus of this work was the development of an efficient Digital Image Correlation algorithm to be integrated in the TopoSEM software package with the same behaviour of the novel MATLAB implementation, real-time computation, and the ability to run efficiently on the computing environments available on the SEM laboratories.

The MATLAB implementation takes 3279,5 seconds to compute SEM images with a resolution of 2576 pixels by 2086 pixels on a quad-core personal laptop, the computing environment most commonly available for researchers. The indicated execution time makes the TopoSEM software package completely unusable in real time. By porting the code to C++ the time to compute the same dataset drooped to 105,01 seconds on the same personal laptop. This represents a speedup of 31,23. Further iterative sequential optimisations focused on the main hotspots of the library lead to a computation time of 3,24 seconds on the same computing environment. Representing a speedup of 31,44 when compared to the initial port and a speedup of 1012.19 when compared to the initial MATLAB implementation. To achieve further optimisation the focus shifted towards parallelization of the code. By creating a generic front wave propagation batching scheduler and applying it to the problem it was possible to achieve a computation time of 1,52 seconds. An overall speedup of 69,09 when compared to the initial C++ port and 2157,57 when compared to the MATLAB implementation on the same hardware.

The library was also tested on homogeneous servers, being able to achieve similar overall speedups in the sequential optimisations and, due to the high core count of this platforms, a lower computation time for the parallel optimisations. This library is capable of achieving a computation time of 0,94 seconds on a 24 core dual-socket server. All this results are summarised in the Figure 29.

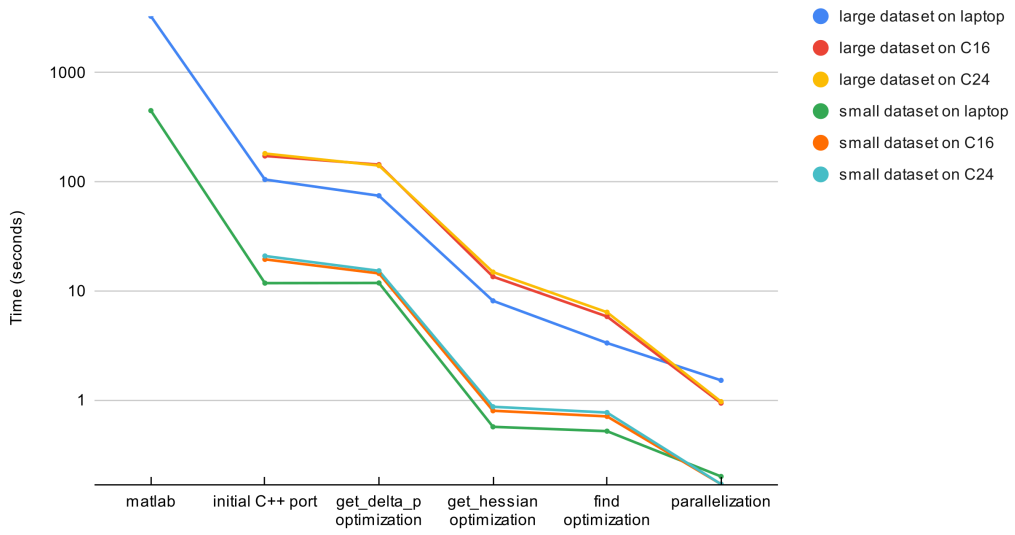


Figure 29: Best execution time (in seconds) for each optimisation step with a logarithmic computation time scale.

In the current version of the DIC algorithm, similar to the MATLAB implementation, the user needs to give the initial guess. As future work, a library or a novel implementation could be integrated into the DIC library to compute the initial guess thus limiting user error and improving the user experience.

The Section 4.2 highlights the limitations of the batching scheduler. A dynamic batching scheduler that varies the size of the batch according to the amount of work in each wave could be developed to mitigate the issues previously highlighted. This way the granularity of the work distribution is maintained for the initial waves and the final waves benefit from the benefits that come from less overhead.

The way the scheduler works could be further altered to support multiple pairs of images with the aim of improving batch processing of pairs of images on servers and clusters. One of the optimisations that could be considered is that instead of sequentially computing the pairs of images all images in the batch would be computed simultaneously. In this way each wave of each image would be simultaneously computed creating more total work for each iteration of the worker threads. Using this method all the negative impact of small initial waves would be mitigated. The cost of creating the thread pool would also be divided across all the pairs of images reducing the cost associated with this step.

In addition, some sections of the DIC algorithm could benefit from GPU acceleration. Namely, the interpolation step that occurs in the pre-computation. The current algorithm used for interpolation is the Bi-cubic since it presents a balance between quality of results and computation speed. After implementing GPU accelerated interpolation the impact of using an algorithm more computationally intensive with improved results, such as Bi-quintic interpolation, could be studied. Although exploring these optimisations could prove beneficial now that the main computation was optimised, they initially only represented less than 0.5% of the overall computation time of the initial C++ port, explaining why there was no focus on optimising this section of the library.

BIBLIOGRAPHY

Bilge Acun, Phil Miller, and Laxmikant V. Kale. Variation among processors under turbo boost in hpc systems. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450343619. doi: 10.1145/2925426.2926289. URL https://doi.org/10.1145/2925426.2926289.

AMD. Amd epyc 9654, 2022. URL https://www.amd.com/en/products/cpu/amd-epyc-9654. [Online; accessed 4-January-2023].

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483485, New York, NY, USA, 1967. Association for Computing Machinery. ISBN 9781450378956. doi: 10.1145/1465482.1465560. URL https://doi.org/10.1145/1465482.1465560.

Analytical Technologies. Technique: 2d-/3d stereo digital image correlation, 2020. URL https://analytical-online.com/digital-image-correlation-dic.html. [Online; accessed 5-December-2022].

Paul E. Anuta. Spatial registration of multispectral and multitemporal digital imagery using fast fourier transform techniques. *IEEE Transactions on Geoscience Electronics*, 8(4):353–368, 1970. doi: 10.1109/TGE.1970.271435.

Luiz F. Bittencourt, Rizos Sakellariou, and Edmundo R. M. Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, PDP '10, page 2734, USA, 2010. IEEE Computer Society. ISBN 9780769539393. doi: 10.1109/PDP.2010.56. URL https://doi.org/10.1109/PDP.2010.56.

J. Blaber, B. Adair, and A. Antoniou. Ncorr: Open-source 2d digital image correlation matlab software. *Experimental Mechanics*, 55(6):1105–1122, Jul 2015. ISSN 1741-2765. doi: 10.1007/s11340-015-0009-1. URL https://doi.org/10.1007/s11340-015-0009-1.

Taisuke Boku. Openmpi-openmp like tool for easy programming in mpi. In *Proc. of 6th European Workshop on OpenMP (EWOMP'04)*, 2004.

Carlos Carvalho. The gap between processor and memory speeds. In *Proc. of IEEE International Conference on Control and Automation*, Jan 2002.

Victor Couty, Jean-François Witz, Pauline Lecomte-Grosbras, Julien Berthe, Eric Deletombe, and Mathias Brieu. Gpucorrel: A gpu accelerated digital image correlation software written in python. *SoftwareX*, 16:100815, 2021. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2021.100815. URL https://www.sciencedirect.com/science/article/pii/S2352711021001102.

L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. doi: 10.1109/99.660313.

John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 6 edition, 2018. ISBN 978-0128119051.

Jetbrains. Developer ecosystem survey 2021, 2021. URL https://www.jetbrains.com/lp/devecosystem-2021/cpp. [Online; accessed 1-February-2022].

Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova. The linux scheduler: A decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342407. doi: 10.1145/2901318.2901326. URL https://doi.org/10.1145/2901318.2901326.

Catalina Mansilla, Václav Ocelík, and Jeff T. M. De Hosson. A new methodology to analyze instabilities in sem imaging. *Microscopy and Microanalysis*, 20(6):16251637, 2014. doi: 10.1017/S1431927614013282.

Frederic Moisy, Marc Rabaud, and E Pinsolle. Measurement by digital image correlation of the topography of a liquid interface. In *ISFV13-13th International Symposium on Flow Visualization, and FLUVISU12-12th French Congress on Visualization in Fluid Mechanics, Paper*, number 326, 2008.

B. Pan, K. Li, and W. Tong. Fast, robust and accurate digital image correlation calculation without redundant computations. *Experimental Mechanics*, 53(7):1277–1289, Sep 2013. ISSN 1741-2765. doi: 10.1007/s11340-013-9717-6. URL https://doi.org/10.1007/s11340-013-9717-6.

Bing Pan, Huimin Xie, and Zhaoyang Wang. Equivalence of digital image correlation criteria for pattern matching. *Applied optics*, 49(28):5501–5509, 2010.

David A Patterson and John L Hennessy. *Computer Organization and Design MIPS Edition: The Hardware/Software Interface*. Morgan kaufmann, 5 edition, Oct 2013. ISBN 978-0124077263.

André Pereira. *HEP-Frame: a Development Aid and Efficient Execution Engine where a Multi-Layer Scheduler Adaptively Orders Pipelined Data Stream Applications*. PhD thesis, Programa de Doutoramento em Informatica (MAP-i) das Universidades do Minho, de Aveiro e do Porto, 2019.

Nadège Van Puymbroeck, Rémi Michel, Renaud Binet, Jean-Philippe Avouac, and Jean Taboury. Measuring earthquakes from optical satellite images. *Appl. Opt.*, 39(20):3486–3494, Jul 2000. doi: 10.1364/AO.39.003486. URL https://opg.optica.org/ao/abstract.cfm?URI=ao-39-20-3486.

A. Radulescu and A.J.C. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, pages 229–238, 2000. doi: 10.1109/HCW.2000.843747.

Hubert W. Schreier, Joachim R. Braasch, and Michael A. Sutton. Systematic errors in digital image correlation caused by intensity interpolation. *Optical Engineering*, 39(11):2915 – 2921, 2000. doi: 10.1117/1.1314593. URL https://doi.org/10.1117/1.1314593.

MA Sutton, Cheng Mingqi, WH Peters, YJ Chao, and SR McNeill. Application of an optimized digital correlation method to planar deformation analysis. *Image and Vision Computing*, 4(3):143–150, 1986. ISSN 0262-8856. doi: https://doi.org/10.1016/0262-8856(86)90057-0. URL https://www.sciencedirect.com/science/article/pii/0262885686900570.

Mullai Thiagu, Sankara J. Subramanian, and Rupesh Nasre. High-speed, two-dimensional digital image correlation algorithm using heterogeneous (cpu-gpu) framework. *Strain*, 56(3):e12342, 2020. doi: https://doi.org/10.1111/str.12342. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/str.12342. e12342 10.1111/str.12342.

Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

Linus Torvalds. The linux kernel mailing list, 2001. URL https://www.tech-insider.org/linux/research/2001/1215.html. [Online; accessed 6-October-2022].

DZ Turner. Digital image correlation engine (dice) reference manual. *Sandia Report, SAND2015-10606 O*, 2015.

Wm A Wulf and Sally A McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors, *Euro-Par 2003 Parallel Processing*, pages 189–194, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45209-6.

Fu Q. Zhong, Padmeya P. Indurkar, and Cheng G. Quan. Three-dimensional digital image correlation with improved efficiency and accuracy. *Measurement*, 128:23–33, 2018. ISSN 0263-2241. doi: https://doi.org/10.1016/j.measurement.2018.06.022. URL https://www.sciencedirect.com/science/article/pii/S0263224118305487.

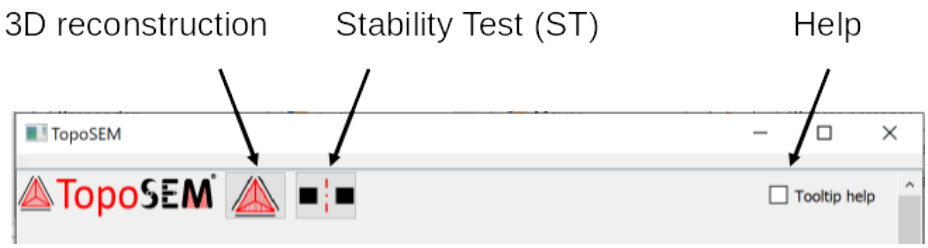
Part I

APPENDICES

A

FEW INSTRUCTIONS TO OPERATE TOPOSEM

In this document, I will try to include some useful tips to run TopoSEM. This is not an exhaustive instructions manual, just a quick user guide. The installation procedure of TopoSEM is not included here, but if you encounter any difficulties, please make a screen capture and/or describe them to us. In here, as its performed by a standard Windows installation wizard. The software has been tested extensively in Windows 10, and should be compatible up to Windows 7. Once open, this window appears:



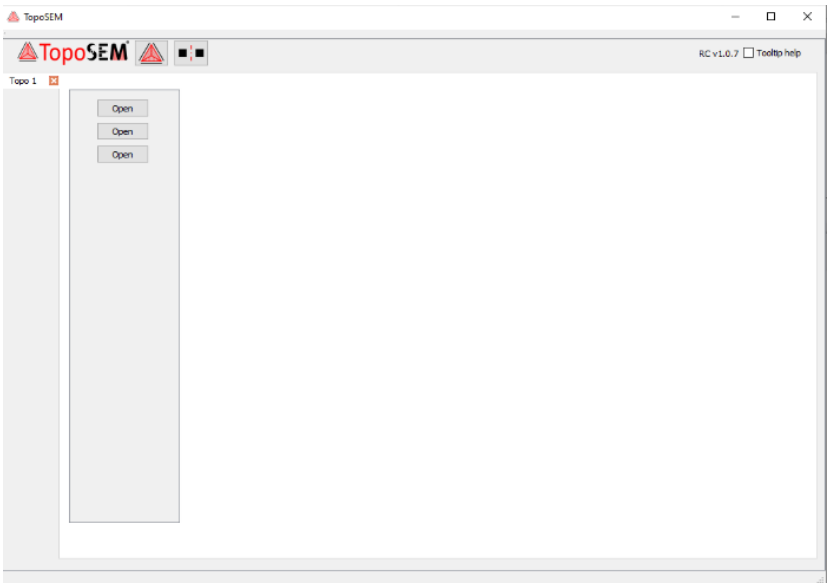
The box on the right activates the tool-tip messages, which are depicted when the mouse hovers over each element and describe its operation and/or meaning. I would recommend its activation for beginner users. This current version of TopoSEM has 2 tools (but it is designed to add easily additional features in future as new icons):

- 3D reconstruction (**Topo**): takes 3 SEM images acquired at 3 different angles and computes the 3D reconstruction.

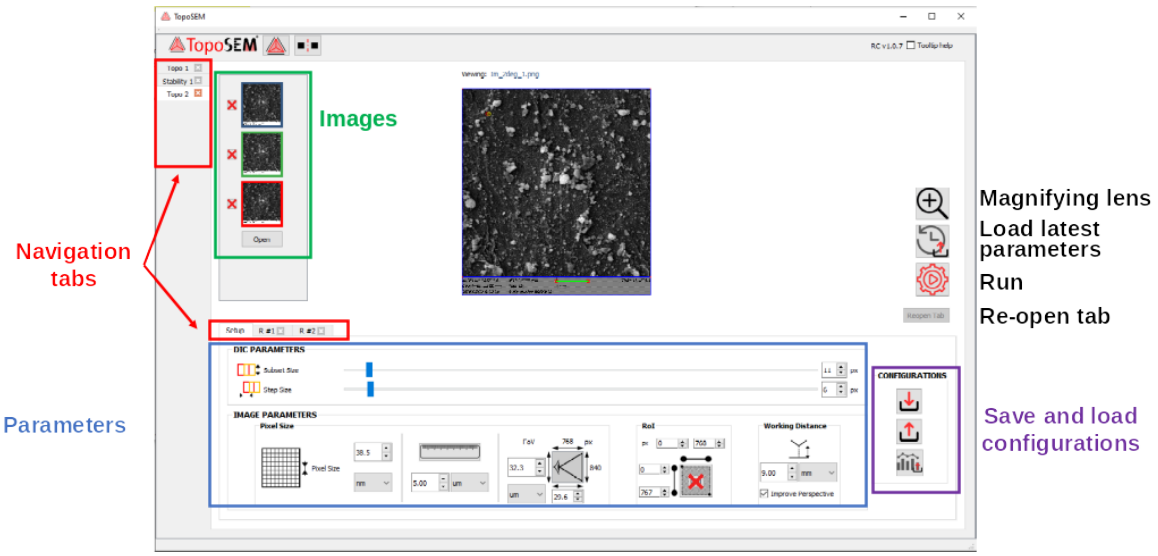
- SEM stability test (**Stability**): takes 2 SEM images acquired in identical conditions and obtains a evaluation of the SEM nonrandom noise during image acquisition.

A.1 PARAMETER INPUT

Both tools work similarly, so I will include just a quick description of the first one. When you press the **Topo** button, you arrive to the following menu:



To load the images the structure of the software is as follows:

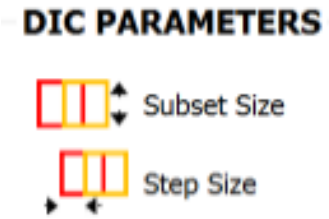


Navigation tabs: TopoSEM is organised with 2 bands of Tabs. The vertical one (left) allows access to all the analyses performed since opening TopoSEM (Topo and Stability). Each tab is created to analyse different group of images, and their deletion cannot be undone. The results obtained for a group of images are accumulated in horizontal tabs. The one on the left (Setup) allows the user to select the images to work and the operation parameters. The other tabs (R#1, R#2, etc.) summarize the results obtained with a certain set of parameters and/or subsets of images. This enables easy comparison of results obtained with different parameters within the same group of images. The Results tabs can be deleted, but they can be recovered using the button Reopen

Tab. While the Setup Tabs are fundamentally identical for Topo and Stability, the Results Tabs are different, and they will be described afterwards.

Images: here the user can select the 3 or 2 images that will be considered for the Analysis (Topo or Stability, respectively). The user can select the hierarchy of the images (i.e. reference, second and third, indicated with frames of that color). This is relevant for Topo, since second and third are compared against the reference and not directly with each other.

Parameters: the user must set several parameters to run the analysis, which are divided into DIC parameters (Digital Image Correlation) and Image parameters.

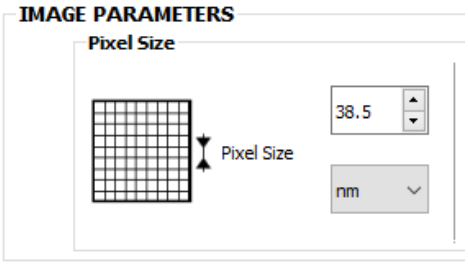


DIC parameters: subset size is automatically pre-selected depending on the resolution of the image, and step size (overlapping) is pre-selected as the half of subset size. The user can adjust these parameters to optimise the analysis. For example, the reduction of the subset size increases the resolution of the result, but it can reduce the accuracy of the subset identification.

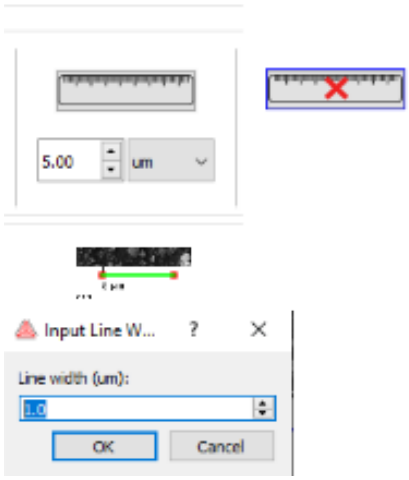
Image parameters: here the user has to introduce 3 parameters connected to the image which are needed to run the analysis: pixel size, RoI (Region of Interest) and Working Distance (this latest one is not used for Stability analyses).

Pixel size: it indicates the dimensions of each pixel. It can be input using three different approaches; when one is used, the values of the other two are updated immediately.

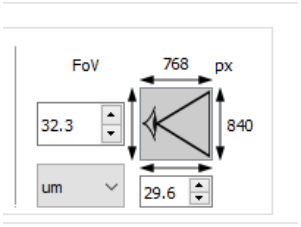
1. Direct input of the pixel size: the user can introduce manually the length units and the size of the pixel.



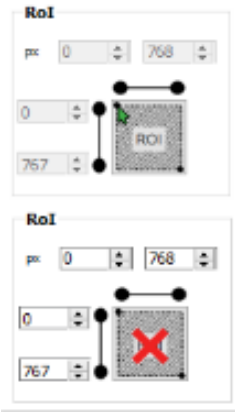
2. with the button set scale: after pressing the icon of the ruler, the user can draw 2 red points (left click) which indicate the left and right limits of the scale bar of the SEM image. As a result, a green bar appears on top of the scale bar, and a pop-up window allows introducing its dimensions (in microns). The scale bar is always horizontal, regardless of the precision of the selection of the second point. The units and dimensions can be modified afterwards using the boxes next to the ruler icon. The scale can be deleted by pressing the ruler icon button again (highlighted with X).



3. by determining the field of view (FoV) of the image. The dimensions of all the image in pixels is shown, and the user can introduce the real size of the image. The dimensions of the image include the scale area (typically on the bottom of the SEM images), so the FoV is normally introduced using the horizontal size.

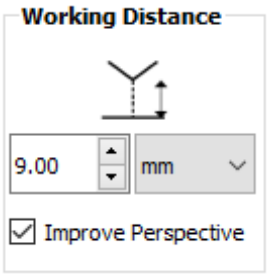


Region of Interest (RoI): it allows the user to exclude from the SEM image the area of the scale bar, or to selectively analyse a particular area of the image. It can be defined by clicking on the button and shifting with the mouse the limits of the image (blue frame). Alternatively, the limits of the RoI can be introduced numerically by selecting the left, right, top and bottom limits. The RoI can be deleted by pressing the RoI icon again (highlighted with a red cross).

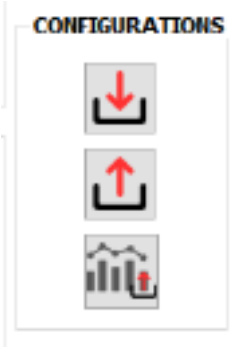


Working distance (WD): this parameter has to be introduced numerically. This is the physical distance from the sample surface to the electron source, and it is normally included in the information of the image next to the

scale bar. Here the user has the option to de-activate the perspective correction (this option is set for debugging, and it will be probably removed in final versions of the software).



Configurations: these three icons allow to use files in the computer to save and load configurations (sets of parameters: DIC parameters, pixel size, RoI, WD). The first and second icons allow to save and load configurations into specific files, while the third one allows to load the configuration from a result previously saved in the computer.



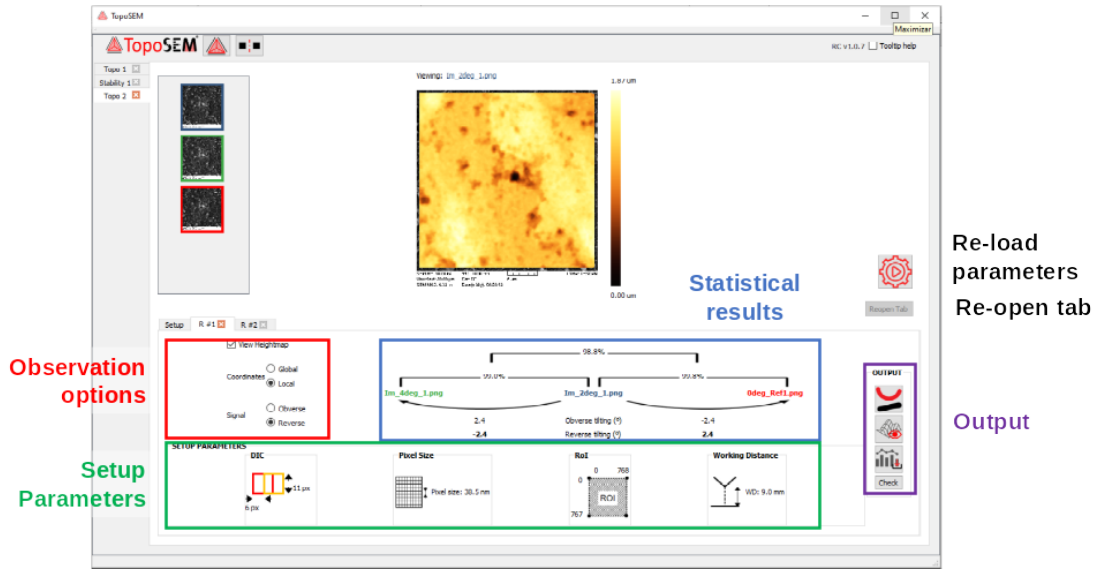
Magnifying lens: this icon opens a new window displaying the SEM image with its original size, where the previous parameters (pixel size, RoI, WD) can be determined with high precision.

Load latest configuration: purely for convenience, this icon can load the latest set of parameters used (DIC parameters, pixel size, RoI, WD). If this is a fresh session, the last set used before closing the last session is re-loaded.

Run: runs Topo or Stability when all the parameters are set.

A.2 3D RECONSTRUCTION VISUALISATION

The results obtained for a 3D reconstruction show the following structure:



The heightmap of the topographic reconstruction (black-orange-yellow-white) is shown overlapping the Reference SEM image.

Observation options: The option View heightmap toggles visibility of the heightmap on top of the SEM image. This option allows the user to check if the features observed in the SEM image coincide with the features observed in the 3D map. There are 4 possible 3D maps, depending on the Coordinates (Local or Global) and the Signal (Obverse or Reverse). Global Coordinates include the tilting of the sample in the SEM chamber, while Local Coordinates represent the sole topography of the surface. The Signal (Obverse or Reverse) represents two possible interpretations of the topographical features: a mountain can be interpreted as a mountain or as a hole. By eye, the user can decide which option is correct (i.e. obverse or reverse).

Statistical results: here we show some numerical results of the calculations. The file name of the Reference image (in blue) is located in the centre, while the names of the Second and Third images are located in the edges. The calculated angles between reference image and the others are indicated below the curved arrows. These values should be similar to the nominal values from the SEM stage holder Highlighted in bold are those corresponding to the selected signal (obverse or reverse). The percentages between the images indicate the % of facets that are recognised in DICs between reference and second (or third) images. The value located in the top indicates the % of facets that are recognised in both DICs. The higher the percentages, the higher number of points used for reconstruction.

Input parameters: the parameters used for this calculation (DIC parameters, pixel size, RoI, and WD) are summarised in this area.

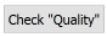
Output: this box summarises additional treatments that can be done to the data.

- **Transfer to WSxM**: this icon opens WSxM and transfers the current 3D reconstruction to WSxM for further analyses. This option is only active for TopoSEM premium users.

- **See 3D topography**: this icon opens a pop-up window with a simple 3D view of the topography of the reconstruction, which can be zoomed and rotated in any direction.

- **Save result into file**: saves the result of the current reconstruction, including the input parameters, in a directory specified by the user.

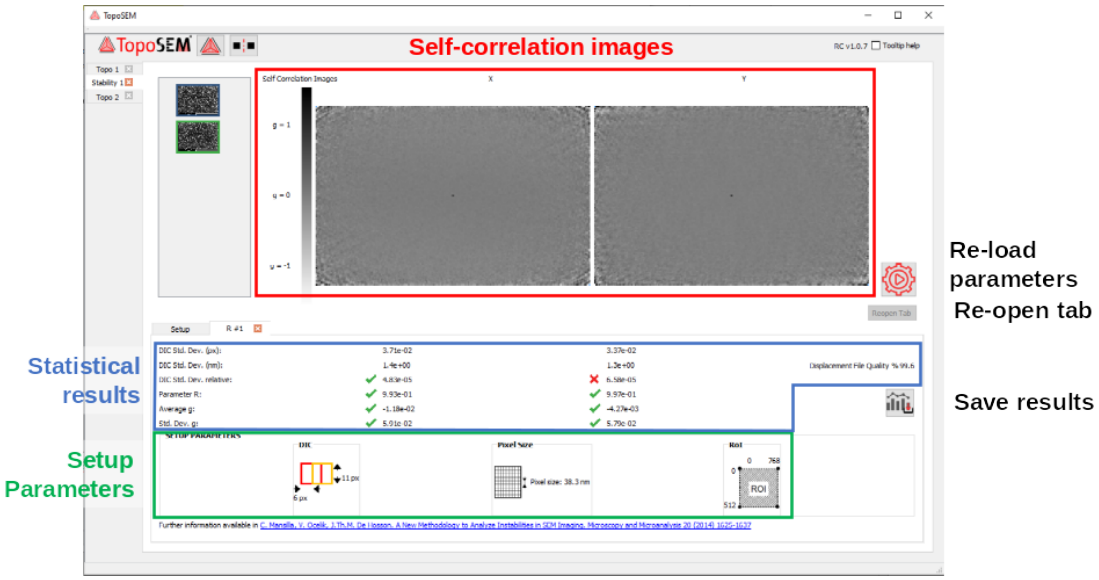
- **Check Quality**: this option is only for debugging, and it will not be shown to conventional users. It represents plots of ddl vs. ddk (see original paper of the method for further information). In general, these plots should show a more-or-less homogeneous and elongated distribution of points.



- **Re-load parameters**: this option loads the analysis parameters (DIC parameters, pixel size, RoI, and WD) and the images used in the reconstructions to the Analysis tab. This option allows the user to recover all the parameters employed to reach a certain result, to easily modify them and re-run an analysis.

A.3 STABILITY VISUALISATION

The results obtained for a Stability Test show the following structure:



This method uses 2 SEM images gathered in succession without any modification of the acquisition parameters, and it is based in a paper whose reference is included in the bottom of the window.

Self-correlation images: from the selected couple of SEM images, two DIC images are obtained, in X and in Y. The self-correlation images are calculated from both DIC images. These images are shown here. Homogeneous grey images indicate lack of non-random noise (i.e. good imaging conditions). In contrast, the appearance of bands indicates the presence of instabilities.

Statistical results: here the output parameters of the method are displaced. If these parameters lay in the acceptance region or not is indicated with a green check mark and a red cross, respectively. The Displacement File Quality indicates the % of the DIC facets that are recognised between SEM both images (it should be close to 100%).

Input parameters: the parameters used for this calculation (DIC parameters, pixel size, and RoI) are summarised in this area.

- **Save result into file**: saves the result of the current result, including the input parameters, in a file in disk.

- **Re-load parameters**: this option loads the analysis parameters (DIC parameters, RoI, pixel size and WD) and the images used in the reconstructions back into the Setup tab. The user may recover all the parameters employed to reach a certain result, in order to make incremental adjustments and improve the analysis.