UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JESSICA IMLAU DAGOSTINI

Performance Improvements Applied in an Electromagnetic Inversion Application Focused on Homogeneous and Heterogeneous Computational Environments

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Lucas Mello Schnorr

Porto Alegre
March 2022

CIP — CATALOGING-IN-PUBLICATION

Imlau Dagostini, Jessica

Performance Improvements Applied in an Electromagnetic Inversion Application Focused on Homogeneous and Heterogeneous Computational Environments / Jessica Imlau Dagostini. – Porto Alegre: PPGC da UFRGS, 2022.

78 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Lucas Mello Schnorr.

1. HPC, mCSEM, MPI, Load Balancing, Oil/Gas. I. Mello Schnorr, Lucas. II. Título.

"Success is to be measured not so much by the position that one has reached in life, as by the obstacles which he has overcome while trying to succeed."

— BOOKER T. WASHINGTON

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Prof. Lucas Mello Schnorr, for the opportunity of working together and all guidance during these past two years. It was not two ordinary years for all of us, where we faced challenges that we never expected to have to do. Even though, we adapt our work to this “new normal” and execute it with our best. I am really glad for all support and knowledge you shared with me during this period of my life.

I would also like to express my gratitude to the Federal University of Rio Grande do Sul (UFRGS) and the Institute of Informatics (INF) for the opportunity of conducting my research within its infrastructure (even no being in person due to the restrictions imposed by the pandemic). It was amazing to have the opportunity to be part of and interact with this scientific community.

I want to express my warm thanks to my parents, Adelar and Elaine, for their support and patience. It was not two regular years for them as well, but they supported me and were my strength in moments where I did not believe in myself. Without them, I wouldn’t be what I am today, and because of them, I’m here. I also want to thank my boyfriend, Jean, for adding to my life during this important period. You were my friend, advisor, and support, and I am really grateful for having you in my life.

I also want to thank my colleagues Ana Solórzano, Valéria Girelli, and Jessica Muller for the fantastic support network we build. I am sure that these two past years would not be the same without our group. Thanks for all meetings where we planned activities to bring more women to our community and for all chatting regarding any other topic. Thanks for also presenting me to the Supercomputing community, where I could learn a lot and discover opportunities that I never even imagined. I can not forget to also thank Vinicius Garcia, that was not only a colleague but an unofficial advisor. All our discussions, your support of my ideas, and all your help were really decisive to the execution of my research.

This research was financed by Petrobras (2018/00263-5) and in part by the Coordenacao de Aperfeicoamento de Pessoal de Nivel Superior - Brasil (CAPES) - Finance Code 001. The experiments were executed in INF/UFRGS’s PCAD, <http://gppd-hpc.inf.ufrgs.br>.

ABSTRACT

Physical oil exploration executed by the oil and gas industry usually requires amounts of money in the order of millions. This industry resorts to numerical and computational methods that help correctly map areas in the ocean with a higher probability of oil and gas incidence and avoid unnecessary drilling operations. One of such methods is the marine Controlled Source Electromagnetic (mCSEM), which uses maritime receivers fixed in a specific region on the seafloor to gather electromagnetic data from such region. An electromagnetic emitter linked to a ship emanates a low-frequency signal and move over this predetermined area. After the electromagnetic waves traverse the subsurface under the seabed and reflect, the receivers measure the field and store this data. We then need an inversion process to recover the resistivity imprinted by the underground materials. Once that oil and gas have known resistivity values, we can better estimate if the region has oil reservoirs and where they are placed. All these processes are computationally expansive, requiring that the application be as optimized as possible to deliver results as fast as possible. This work studies an inversion application that uses finite direct equations and the common mid-point technique to deal with data collected from the mCSEM method. This application has three main steps: forward, linear system solution, and correction, where the first is parallelized to deliver better performance. We found space for computational improvements in our application with a load imbalance analysis once we identified an issue from the original workload scheduling policy. We implement and evaluate load balancing heuristics to improve the scalability in different cluster configurations in homogeneous and heterogeneous environments. We improved 44% in homogeneous environments and 78% in heterogeneous environments with our implemented solutions compared against the original. Furthermore, we develop a capacity planning for this application. We designed a tool that uses traces from previous controlled executions with a specific study case and predicts the application's behavior to a set of given machine combinations. Such a tool helps in this planning by allowing users to manipulate this simulated data using interactive visualization.

Keywords: HPC, mCSEM, MPI, Load Balancing, Oil/Gas.

Melhorias de Desempenho com Foco em Ambientes Computacionais Homogêneos e Heterogêneos Aplicadas em uma Aplicação de Inversão Eletromagnética

RESUMO

A exploração física de petróleo executada pela indústria de petróleo e gás costuma demandar valores da ordem de milhões de dólares. Esta indústria então recorre a métodos numéricos e computacionais para ajudar a mapear corretamente as áreas oceânicas com maior probabilidade de incidência de petróleo e gás e evitar operações de perfuração desnecessárias. Um desses métodos é o *marine Controlled Source Electromagnetic* (mCSEM), em Português, Método Eletromagnético Marinho de Fonte Controlada. Este método utiliza receptores marítimos fixados em uma região específica do fundo do mar para coletar dados eletromagnéticos dessa região. Um emissor eletromagnético ligado a um navio se move sob esses receptores e emana um sinal eletromagnético de baixa frequência. Depois que as ondas eletromagnéticas atravessam a subsuperfície sob o fundo do mar e refletem, os receptores medem o campo em questão e armazenam esses dados. Precisamos então de um processo de inversão de dados para recuperar a resistividade impressa pelos materiais subterrâneos. Uma vez que o petróleo e o gás possuem valores de resistividade conhecidos, podemos estimar melhor se a região possui reservatórios deste materiais e onde eles estão localizados. Todos esses processos são computacionalmente caros, exigindo que a aplicação seja o mais otimizada possível para entregar resultados o mais rápido possível. Este trabalho estuda uma aplicação de inversão de dados que utiliza equações diretas finitas e a técnica do ponto médio comum, do inglês *Common Mid-Point*, para lidar com dados coletados a partir do método mCSEM. Esta aplicação possui três etapas principais: modelagem direta, resolução do sistema linear e correção, onde a primeira é paralelizada para oferecer melhor desempenho. Encontramos espaço para melhorias computacionais em nossa aplicação com uma análise de desequilíbrio de carga, uma vez que identificamos um problema na política original de distribuição de carga. Implementamos e avaliamos heurísticas de balanceamento de carga para melhorar a escalabilidade em clusters homogêneos e heterogêneos. Melhoramos o tempo de execução da aplicação em 44% em ambientes homogêneos e em 78% em ambientes heterogêneos. Além disso, desenvolvemos um planejamento de capacidade de execução, do inglês *capacity planning*, para esta aplicação. Projetamos uma ferramenta que utiliza rastros de execuções controladas anteriores com um caso de estudo específico para prever o comportamento da aplicação para um conjunto determinado de combinações de máquinas. Tal ferramenta auxilia neste planejamento ao permitir que os usuários manipulem esses dados simulados utilizando uma visualização interativa.

Palavras-chave: HPC, mCSEM, MPI, Balanceamento de Carga, Petróleo.

LIST OF ABBREVIATIONS AND ACRONYMS

ABE	Area-Bound Estimation
ANOVA	Analysis Of Variance
CMP	Commom Mid-Point
CPU	Computing Process Unit
CSV	Comma-separated values
EM	Electromagnetic
FDFD	Finite-Difference Frequency-Domain
GPU	Graphic Process Unit
HEFT	Heterogeneous Earliest Finish Time
HPC	High Performance Computing
mCSEM	marine Controlled Source Electromagnetic
MPI	Message Passing Interface
ODE	Ordinary Differential Equations
PCAD	Parque Computational de Alto Desempenho
PCAM	Partitioning, Communication, Agglomerations, and Mapping
PDE	Partial Differential Equations
PSNR	Peak Signal-To-Noise Ratio
S-R	Source-Receiver Pairs

LIST OF FIGURES

Figure 1.1 Example of an imbalanced execution with traces collected from our experiments with the application studied in this work. 15

Figure 2.1 A synthetic load characterizing number partitioning. 21
Figure 2.2 A synthetic distribution with the Round Robin heuristic. 21
Figure 2.3 An homogeneous load distributed with the Round Robin heuristic. 22
Figure 2.4 Synthetic distribution with Sorted-Greedy approach. 22
Figure 2.5 A synthetic distribution with HEFT approach. 24

Figure 3.1 The mCSEM exploration. 26
Figure 3.2 3D synthetic model with 1.5 kilometer of water layer. Target regions are in red. 27
Figure 3.3 Survey data (red triangles are receivers, small blue squares are emitters), combined with the 2D grid of CMPs (circles). 28
Figure 3.4 Example of an output model after the inversion process. It is an slice at the depth of 1614 meters collected from iteration 99 of the application. 29
Figure 3.5 Data inversion with mCSEM. 29
Figure 3.6 Workflow diagram of the application in study. 30
Figure 3.7 A synthetic load characterizing the behavior of the original distribution process. 31

Figure 5.1 Methodology steps executed in this contribution. 38
Figure 5.2 Number of S-R pairs associated for each CMP of the studied model. 40
Figure 5.3 Temporal data collected in one execution with three iterations. Colors represent different operations detected during execution. 40
Figure 5.4 Execution behavior of the application executing only with MPI, presenting the timespace of the three main regions of the inversion algorithm. 41
Figure 5.5 Behavior of the original distribution – it divide the cells among workers, disregarding its own heterogeneous weight. 42
Figure 5.6 The number of S-R combinations explains processing time. 42

Figure 6.1 Methodology applied to this contribution. We used the intrumentation made previously to analyze the results provided from our distribution algorithms. 45
Figure 6.2 Theoretical distribution of load with 1 up to the number of CMP cells of the study case, presenting the distance from ideal mean from these settings (top) and the maximum load allocated to some worker (bottom). 47
Figure 6.3 The distance from ideal load balance for every worker considering five total number of workers (columns) and the CMP scheduling policies (rows). 48
Figure 6.4 Space/time view of the forward step of 100 iterations for the Original, Round-robin, and Greedy allocation policies. 50
Figure 6.5 Speedup of 1 up to 112 MPI processes. We run three iterations of the application. The baseline is Original with one process. The dashed black line represents the ideal speedup. 51
Figure 6.6 Speedup of many MPI processes from 1 up to 212 using two different partitions with the same processor's frequency. We also run the application with three iterations, and the baseline is Original with one process. Dashed black line represents the ideal speedup. 52
Figure 6.7 Average time of execution of one iteration in each scheduler running on different heterogeneous configurations. 55

Figure 6.8 The distance from ideal load balance for every worker considering the three different heterogeneous scenarios (columns) and the new scheduling policies (rows)....56
Figure 6.9 Average makespan with execution of 100 iterations for each scheduler in two different heterogeneous scenarios....57
Figure 6.10 Model output from all schedulers....58

Figure 7.1 Figure with methodology....60
Figure 7.2 Workflow diagram of the simulator developed for the capacity planning....61
Figure 7.3 Screen capture of the interactive plot with all possible scenarios of machine's combination. The colors of the plot represent the different algorithms, and the different size of points indicates how many cores each scenario is composed of....62
Figure 7.4 Validation of results from our R simulator code. We validate our simulation with HEFT algorithm. The blue points represent the average runtime of real executions....63
Figure 7.5 Makespan with combination scenarios involving different quantities of Optimizer machines running HEFT distribution heuristic....64
Figure 7.6 Examples of CMP allocation in two scenarios, one using two Optimizer machines and other using four new machines. We can identify that, when using four machines, some workers did not receive any load to compute....65

Figura A.1 Exemplo de execução com distribuição desbalanceada, gerado a partir de dados coletados da execução da aplicação em estudo....75

LIST OF TABLES

Table 6.1 Specification of machines used in the executions. 46
Table 6.2 Setting of the three different case studies for algorithms comparison. 55

CONTENTS

1 INTRODUCTION **13**
1.1 Motivation **14**
1.2 Contributions **16**
1.3 Document Structure **17**
2 BACKGROUND **18**
2.1 Parallel Environments **18**
2.2 Load Balancing **19**
2.2.1 Round Robin 21
2.2.2 Sorted-Greedy 22
2.2.3 Area Bound Estimation - Guided Approach 23
2.2.4 Heterogeneous Earliest Finish Time 23
3 THE MCSEM INVERSION APPLICATION **25**
3.1 mCSEM Method **25**
3.2 The model **26**
3.3 Inversion Process **28**
3.4 Parallelization **30**
4 RELATED WORK **33**
4.1 Algorithm and Numerical Improvements Focus **33**
4.2 Computational Improvements Focus **35**
4.3 Discussion **36**
5 CONTRIBUTION: FINDING THE IMBALANCE **38**
5.1 Methodology **38**
5.2 Mapping Data to the Model **39**
5.3 Problem Identification **40**
5.4 Final Remarks **43**
6 CONTRIBUTION: APPLYING HEURISTICS TO THE PROBLEM **44**
6.1 Materials and Methods **44**
6.2 Theoretical Analysis on a Homogeneous Environment **46**
6.3 Heuristics in Homogeneous Environments **47**
6.4 Heuristics in Heterogeneous Environments **51**
6.5 Numerical Accuracy **57**
7 CONTRIBUTION: CAPACITY PLANNING **59**
7.1 Materials and Methods **59**
7.2 Simulating Real Executions **60**
7.3 Simulation Validation **62**
7.4 Hypothetical Test **63**
7.5 Final Remarks **64**
8 CONCLUSION **66**
8.1 Publications **67**
REFERENCES **68**
APÊNDICE A — RESUMO EXPANDIDO EM PORTUGUÊS **73**
A.1 Motivação **73**
A.2 Contribuições **76**
A.3 Conclusões **77**

1 INTRODUCTION

One of the main challenges in the oil and gas industry is to correctly map areas in the ocean where there is a higher probability of oil and gas incidence. Physical oil exploration usually requires amounts of money in the order of millions of dollars. To help on this task, scientists and companies recur to computational simulations to reduce the incidence of unsuccessful drilling operations, thus saving time and money. Various numerical and computational methods can be applied to this exploration involving electromagnetic and seismic data.

One of the electromagnetic surveys to simulate and computationally investigate ocean lands is the marine Controlled Source Electromagnetic (mCSEM) (SOUZA, 2007). This method uses maritime receivers fixed in a specific region of the ocean floor to gather electromagnetic data from such regions. A ship carrying an electromagnetic emitter moves in this predetermined area, and the receivers measure the field after the electromagnetic waves traverse the subsurface under the seabed and reflect. This electromagnetic surveying is applied to recognize the seafloor's aspects, thus giving relevant information about the resistivity of the explored region.

To interpret this electromagnetic data, we need to use numerical methods to apply a mathematical inversion to approximate a matching subsurface model quickly and, this way, have a better comprehension of it. Such subsurface model is referenced here as the resistivity model and has an initial guess of the resistivities of the region in analysis. The data inversion technique is commonly used in oil/gas computational applications since oil and gas have known signatures compared to other materials (KEY, 2009).

We studied an inversion application that uses direct equations and the common mid-point technique to deal with data collected from the mCSEM method. This application maps data from the survey to the initial resistivity model's points in the initialization. Then, in a forward phase, the application runs numerical equations based on Maxwell and Levenberg-Marquardt methods to evolve the model iteratively. These equations are the most expensive step of the inversion process once it performs most of all numerical equations of the method. To run such an application in a reasonable time, we need to split the load of the most expensive steps of it among parallel workers.

High-performance clusters are crucial to the processing speeds required from these significant volumes of data and processes. In these environments, we use machines that have a powerful process to run expensive applications. However, only using these environments is not enough to mitigate computational costs. We need to use cluster machines as best as possible, guaranteeing that all resources are being used at their maximum and balanced. Each resource must receive enough load to compute in a similar duration. This distribution gave the application a load balance characteristic, one key point to its speed.

We found space for improvements in our application with a load imbalance analysis. We identify that the issue comes from the original workload scheduling policy, directly impacting the parallelized forward phase. We then implement and evaluate load balancing heuristics to improve the application, considering the necessity of distributing such data according to the resistivity map given. We aim to enhance the application speed by better using the available resources and improving its scalability in different cluster configurations in homogeneous and heterogeneous environments. Furthermore, we develop capacity planning for this application. We designed a tool that uses traces from previous controlled executions with a specific study case and predicts the application's behavior to a set of given machine combinations. Such a tool helps in this planning by allowing users to manipulate this simulated data using interactive visualization.

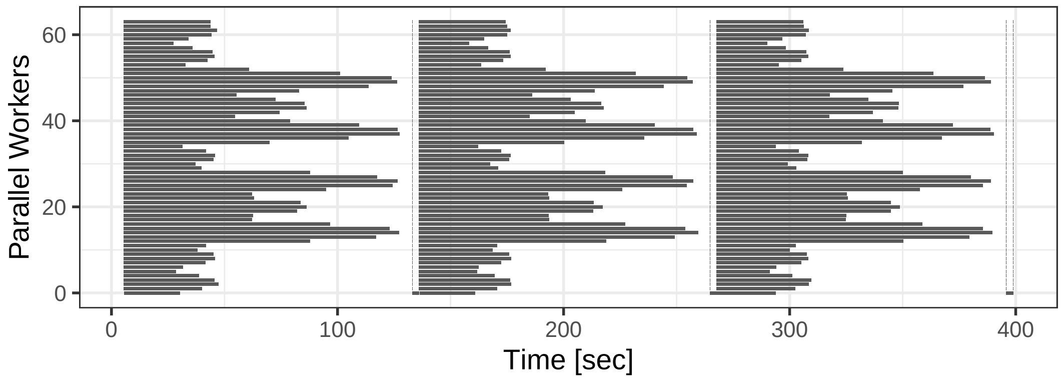
1.1 Motivation

A computational perspective in any scientific application is always essential. There is no complex area of science that does not use computers for modeling (GOLUB; ORTEGA, 1993). This way, it is essential that not only mathematicians correctly model the numerical equations – needed to solve each specific problem – but also that computer scientists model the application according to their knowledge regarding performance and computability. We aim to bring a computational perspective by proposing and executing a practical performance evaluation using state-of-the-art tools for the mCSEM inversion application studied.

Load balance techniques guarantee the efficient resource usage on high performance applications and increase applications' speed and scalability to mitigate this issue. Guarantee such efficiency is critical for today's parallel applications, once it can mean that thousands of cores are allocated to executions but can be idle (PEARCE et al., 2012). There are a considerable number of load balancing strategies studied in the literature. Applying such load balancing solutions to real-world problems can contribute to the evolution of this research area.

Search for a load imbalance on an application requires instrumentation of the code and place time marks to know how long each parallel worker is taken to execute a task. When the application's traces show that some workers take more time to complete a task than others, we have a load imbalance. Figure 1.1 demonstrates this concept with our application scenario. We depict the results of the first three iterations out of a run with 100 iterations, presenting the execution time in seconds on the X-axis and the parallel workers on the Y-axis. In these traces, the first worker finishes their tasks $2.3\times$ faster than the nineteenth worker. These results motivate us to apply computational improvements to our investigated application.

Figure 1.1 – Example of an imbalanced execution with traces collected from our experiments with the application studied in this work.



Source: Author

Improvements on mCSEM applications have some other reports in the literature. (ZHAO; YU; QIN, 2018) and (SILVA et al., 2012) presented works regarding numerical improvements on the inversion process. The first applied a total-field algorithm to the inversion process, using a 3D layered model and the METIS library for the domain partitioning. The second divided the inversion application into two components and used a multilevel partitioning method, applying the MUMPS library for parallelization and a hybrid MPI+OpenMP code. Otherwise, (LONG et al., 2020) and (da Piedade et al., 2021) discuss the benefits of using the Pardiso library to achieve computational speed gains. Paradiso is a software for solving large sparse linear systems on shared and distributed-memory (ALAPPAT et al., 2020). However, no work focused on load balancing strategies for improving performance. We applied state-of-the-art load balancing heuristics in our work and proposed a capacity planning with such strategies.

1.2 Contributions

This research proposes and executes performance analysis and improvement of a real-world mCSEM inversion application. We search for possible computational bottlenecks that originate an imbalanced execution in the forward step. We found out that the strategy initially used to distribute the load among workers disregards the actual load of the grain to distribute. We demonstrate that this load is static and irregular, presenting a similar duration throughout iterations and not showing load difference across the application's execution.

We then proposed load balance improvements regarding homogeneous and heterogeneous environments and applied state-of-the-art solutions to mitigate it with minor changes to existing code. We find four heuristics that we applied to our problem: a Round Robin, Sorted-Greedy, Area Bound Estimation based solution, and Heterogeneous Earliest Finish Time approaches. We adapted such heuristics to our needs and execute tests to validate the performance benefits.

Furthermore, due to the static characteristic of the input load in each environment where it was executed, we also propose a capacity planning to help in the execution's plan in larger clusters. We developed a feature that helps determine how much time the application will take to execute in a diverse set of resource combinations. Such a feature brings a versatile and decisive step in the application's running, allowing us to anticipate how our execution will behave in a determined setup. Such predictions support a better scheduling of resources.

In summary, these are the main contributions:

- Proposal of a practical performance analysis, with a deeper investigation of the application's code and its instrumentation and the identification of the main regions of the application according to the numerical method that led us to identify a load imbalance in such application and its reasons (Chapter 5).
- Study and implementation of load balance heuristics to homogeneous and heterogeneous environments, expanding execution possibilities in different scenarios. where we achieved a performance improvement of 44% in homogeneous clusters and 78% in heterogeneous clusters (Chapter 6).
- Development of a capacity planning to such application, based on the input data and the target platform where it can be executed (Chapter 7).

1.3 Document Structure

This document is organized as follows. Chapter 2 introduces some important concepts regarding parallel programming and load balancing, and presenting the five heuristics applied in this work. Chapter 3 presents the marine Controlled-Source application, discussing its numerical method, the characteristics of its input data, how the inversion process occurs, and how this application is parallelized. Chapter 4 discusses related work regarding improvements on other existing mCSEM applications and their characteristics. We then start talking about our contributions in Chapter 5, demonstrating how our practical performance analysis led us to find the imbalance in the application. We then follow to present the results of the load balance heuristics in homogeneous and heterogeneous environments in Chapter 6. We present our capacity planning as our last contribution in Chapter 7. Finally, in Chapter 8 we conclude this work and discuss future work.

2 BACKGROUND

Parallel programming is a crucial approach nowadays. We have improved artifacts that collect much more data to be correctly synthesized to give information. We need as much processing power as possible to deal with all this data and use multiple resources to achieve it. Many applications use parallel processes to achieve better performance, mainly those that perform a considerable quantity of numerical computation.

However, we do not need to only care about using parallel mechanisms on applications, but it is also essential to guarantee that such parallelization efficiently uses all available resources. The application's strategy to distribute the load among the parallel workers is vital in achieving this efficient resource usage. It directly determines if all resources will take similar times to execute each task and deliver a faster application.

In this chapter, we present a background of parallelism and load balancing, highlighting some important concepts, such as parallel programming levels and different parallel environments. Section 2.1 presents fundamental concepts regarding parallelism, followed by Section 2.2 where we discuss concepts regarding load balancing. Section 2.2 will also present the five load balancing heuristics applied in this work.

2.1 Parallel Environments

Parallelism can be achieved at the programming level as also as hardware level. We can exploit different models on parallel programming, such as in a thread-level and a process-level (RAUBER; RÜNGER, 2013). When working with parallel applications, we aim to break its execution into small pieces that different cores can execute. One famous methodology to parallelize an application is the PCAM, acronym for Partitioning, Communication, Agglomerations, and Mapping (FOSTER; FOSTER, 1995). In an ideal scenario, we hope that a parallelized application with PCAM uses an undetermined quantity of process units efficiently.

There are two main programming models that we can use to run applications in parallel. The first is the Shared-Memory model, where we assume that the parallel resources are using the same memory device, i.e., using a single node. It must provide ways for starting up threads, assigning work to them, and coordinating their accesses to shared data, including ensuring that certain operations are executed by only one thread at a time, without racing conditions (TROBEC et al., 2018). OpenMP and Pthreads are parallel programming interfaces that implement the Shared-Memory model.

The second model uses distributed memory machines, where we need to use a message-passing interface to communicate among the processes. This model assumes that the program runs in resources that have their own private memory space, thus not accessing the information in a shared space. Message-passing approaches must provide a mechanism to initiate and manage all procedures and send and receive messages and data among the different processes. The community developed the MPI standard, which offers a set of more than 120 operations for parallel programming (TROBEC et al., 2018). Open-MPI and MPICH are examples of interfaces that implement this programming model.

Alongside these models, we have different machine clusters to execute applications in parallel. In homogeneous environments, every machine on a cluster has the same settings and speed. They are all built with the same processor unity, memory, and communication settings. On the other hand, a heterogeneous system is composed of machines with different characteristics. Nowadays, most Top500 supercomputers employ an intra-node heterogeneity in their configurations, once each cluster node contains at least one type of accelerator (DONGARRA et al., 2017). We can also have a cluster with system-level heterogeneity, where different computational node types can exist. Typically, we characterize such heterogeneity by differences in the design of the machines, where they have different process units, memory size, and speed and communications buses (NESI; LEGRAND; SCHNORR, 2021).

However, none of these parallel paradigms or types of clusters are enough to guarantee that an application will be improved when parallelized. We must pay attention to how efficient is the usage of our resources by our parallel applications, mainly when splitting an application's load. The correct load distribution ensures that all workers are taking a similar time to compute without any resource slowing down the application. Determine the efficiency of this distribution is a crucial metric to consider in parallel applications.

2.2 Load Balancing

The capability of a parallel program to compute equally in all resources is what we call load balance (ALAKEEL; ALAKEEL, 2010). A load of an application can be classified as static or dynamic and as regular or irregular. A static load does not change during the application's execution, while dynamic load means changing on execution. When an application's load present both static and irregular characteristics, we say that the application has a spatial load imbalance. In this scenario, we can use static solutions executed prior to the actual execution of the application. Otherwise, when the application is dynamic and irregular, we say that it suffers from temporal load imbalance and that it is necessary to apply adaptative solutions to mitigate it (TESSER, 2018).

There are many details to take care of when trying to achieve a balanced application. We need to understand if the workload characteristics will be a complex or straightforward load grain (composed of different structures that can dictate its weight). We also need to know the environment characteristics where we aim to execute the application – i.e. if the machines have the same or different settings or speeds. We have different approaches to achieve load balance according to this set of configurations.

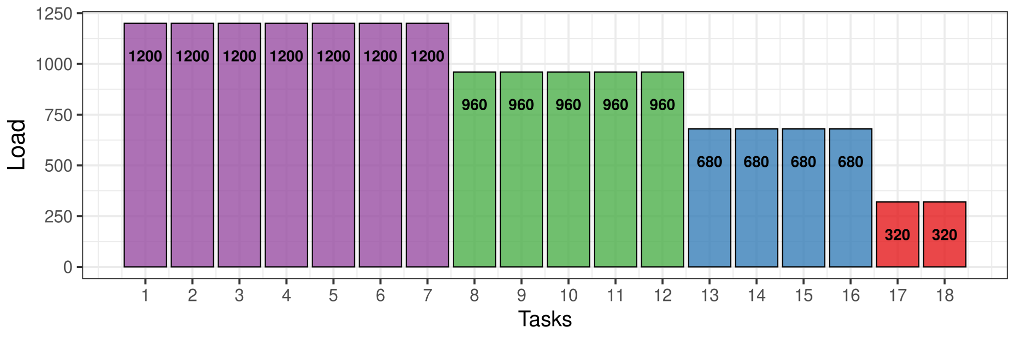
To know how much we could improve an application by running it in parallel, we need to gather metrics and analyze its results. Instrumentation of code is a technique to get this information. We define regions on the program to capture the time it starts and ends to execute. This data can then be analyzed and determine some application's metrics, such as its makespan and speedup, alongside the load balance. The former measure indicates how much time an application takes to execute from its beginning until its end. The second determines a factor by which the parallel program's execution time is faster than the sequential program. It is based on Amdahl's Law, which states that the overall application's performance improvement is limited by the fraction of the code that can be parallelized (CASANOVA; LEGRAND; ROBERT, 2008). The following formula formally describes the speedup:

$$S_p(n) = \frac{T_{seq}(n)}{T_{par}(p,n)} \tag{2.1}$$

where n is the size of the problem, and $T_{seq}(n)$ and $T_{par}(p,n)$ are the makespan in sequential and parallel environments, respectively.

In the context of this work, we will focus on load balancing strategies to spatial load imbalances. We will use the synthetic load depicted by Figure 2.1 to help with the explanations. Here we have a heterogeneous load, composed of eighteen tasks with different sizes, from 320 to 1200 units. Our goal is to distribute this load among different workers without splitting each task. This problem can increase when we consider distributing the tasks among heterogeneous workers. For this, we need to consider the tasks heterogeneity and the different worker's capability. Both heterogeneous characteristics increase the complexity of the balancing algorithms once we have two heterogeneous levels to take care of. To solve these scenarios, we need to use different strategies. With all the former reasons, we will better describe four distribution heuristics that we applied in this work: Round Robin, Sorted-Greedy, Area-Bound Estimation (ABE), and the Heterogeneous Earliest Finish Time (HEFT).

Figure 2.1 – A synthetic load characterizing number partitioning.

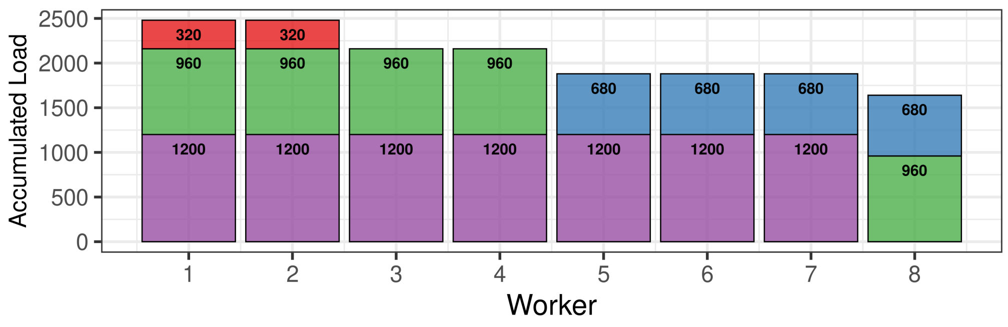


Source: Author

2.2.1 Round Robin

Round Robin is a well-known scheduler algorithm that considers the load as homogeneous to distribute in an offline mode. The algorithm's behavior consists of assigning load among the workers in circular order, from the first to the last worker, disregarding the load already assigned to them. The time complexity of this algorithm is $O(n)$. Figure 2.2 depicts the distribution of our synthetic load with Round-robin. The X-axis represents the workers involved in the simulation, and the Y-axis presents the accumulated load for each worker.

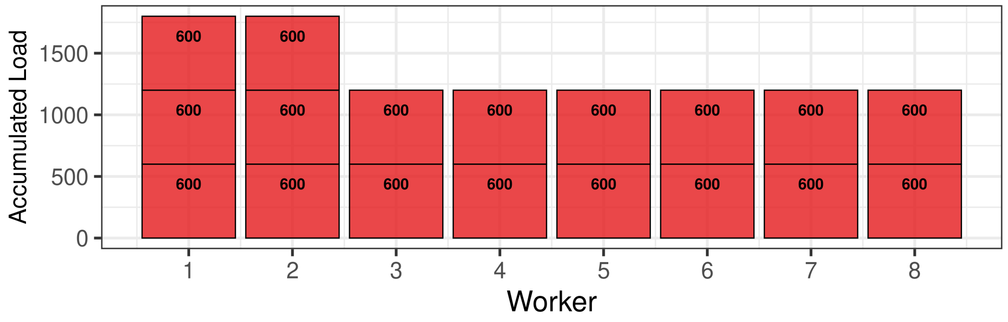
Figure 2.2 – A synthetic distribution with the Round Robin heuristic.



Source: Author

Even presenting a more balanced distribution among workers, we can still identify load imbalance in this algorithm to heterogeneous load such as the one we simulate. Round-robin is well effective with a homogeneous load, as a second example depicted by Figure 2.3 demonstrates.

Figure 2.3 – An homogeneous load distributed with the Round Robin heuristic.



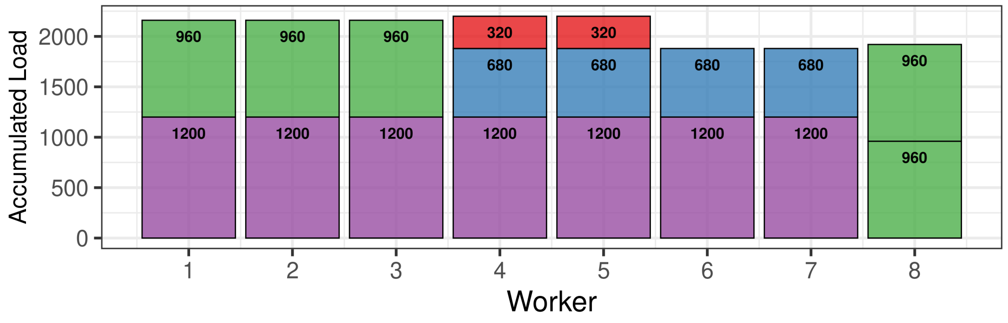
Source: Author

2.2.2 Sorted-Greedy

Sorted-Greedy algorithm is a heuristic used to solve problems of independent tasks that derive from number partitioning problems (MERTENS, 2006; GAREY; JOHNSON, 1979). It considers that every worker has the same capacity and speed but considers the load already assigned to a worker before assigning a new task. For such characteristics, this heuristic delivers a better distribution of heterogeneous load.

This algorithm schedules, at each step, the current task/load to the least-loaded worker, where tasks are sorted in decreasing weight. To perform this approach, we need to know all tasks/loads to schedule before starting the execution. For this reason, Sorted-Greedy is called an offline algorithm. If we have five tasks, three of weight two and two of weight three to be divided between two workers, Sorted-Greedy has a makespan of 7, while optimal is 6, thus given a $\frac{7}{6}$ approximation (CASANOVA; LEGRAND; ROBERT, 2008). Figure 2.4 demonstrates the behavior of this algorithm using the synthetical load previously described. The algorithm will first distribute the load among all free workers (first blocks in each worker). With no more free workers, the algorithm will select the one with less load assigned, which will be worker 8 in this example. It will assign the

Figure 2.4 – Synthetic distribution with Sorted-Greedy approach.



Source: Author

task/load to this worker and go to the next, repeating it until having no more tasks to distribute. This algorithm runs in $O(n \log n)$ time.

2.2.3 Area Bound Estimation - Guided Approach

The Area Bound Estimation (ABE) is a linear programming applied by (PINTO, 2018) to estimate the makespan for task-based applications in heterogeneous applications (that uses CPU and GPUs). It does not consider dependencies among tasks/load. The following equations define our linear programming

Minimize: (2.2)

$$makespan$$

subject to:

$$\forall t \in T : \quad \sum_r n_{t,r} = N_t \tag{2.3}$$

$$\forall r \in R : \quad \sum_t n_{t,r} * w_{t,r} \leq makespan * N_r \tag{2.4}$$

$$\forall t \in T, r \in R : \quad n_{t,r} \geq 0 \tag{2.5}$$

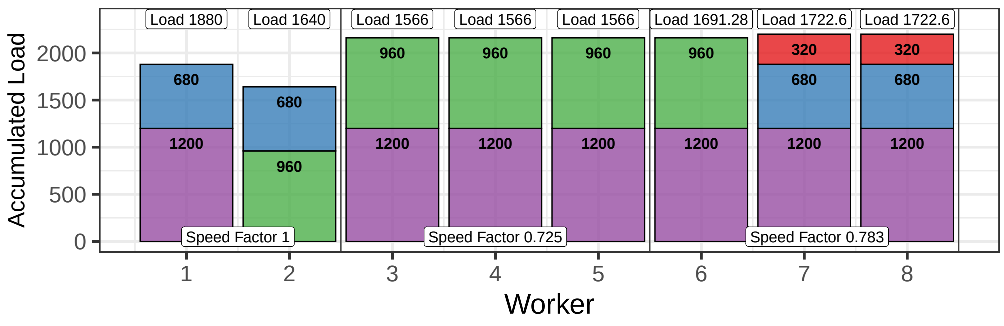
where t is a determined group of tasks T with the same size, r is the type of resource R (that are the different machines where we are executing the code), and $w_{t,r}$ is the meantime which the determined group of tasks t takes to execute in a determined resource. Such linear programming provides the best makespan for each heterogeneous scenario and the ideal allocation of each different task size to each resource that generates the makespan.

2.2.4 Heterogeneous Earliest Finish Time

The Heterogeneous Earliest Finish Time (HEFT) is also an algorithm applied to the task scheduling problem. Its original implementation consists of a task prioritizing phase, using a previous application benchmark in each available resource, and a processor selection phase to schedule the tasks on its best processor (TOPCUOGLU; HARIRI; WU, 2002). This benchmark consists of executing the application in a way that we can gather traces to use as a baseline of the application's behavior in a specific resource. On the task prioritizing phase, this priority is based on each task's average computation and average communication costs and its precedence constraints. The HEFT algorithm applies an insertion-based policy that considers the earliest idle time slot available to schedule a task to a processor in the selection phase. This heuristic iterates throughout all tasks, testing the task insertion in all available resources, which gives us $O(t \times r)$ time complexity.

Considering its description, Figure 2.5 presents results of applying HEFT heuristic in our synthetic example, using the same synthetic data former presented. The two first workers present a speed factor of 1, workers 3, 4, and 5, a speed factor of 1.725, and workers 6, 7, and 8 a speed factor of 1.783. The first workers received less load to compute due to its low-speed factor, while the last workers had the most load due to its best characteristics.

Figure 2.5 – A synthetic distribution with HEFT approach.



Source: Author

3 THE MCSEM INVERSION APPLICATION

Electromagnetic surveying is employed in both land and maritime contexts – though these have developed almost independently, either actively or passively (CONSTABLE; SRNKA, 2007). This survey generates a collection of electromagnetic data used to identify different materials according to their known resistivities. The underground materials imprinted their different conductivities onto the now-read electromagnetic field values. Therefore, mCSEM applications aim to model back the resistivities of these materials from the read values through an inversion method, matching the readings to the most probable model of the subsurface substances that generated such readings.

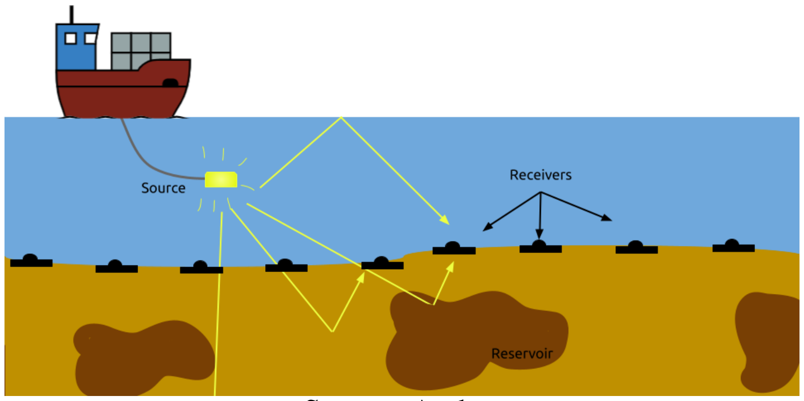
In the next sections, we introduce how this inversion process happens in the studied application at this work. We start giving an overview of the marine Controlled Source Electromagnetic (mCSEM) method in Section 3.1. We follow by understanding the model that determines all the inversion processes in Section 3.2. Section 3.3 details the inversion process and how the application executes the numerical processes. Finally, Section 3.4 reports the application's organization and parallelization.

3.1 mCSEM Method

In the marine controlled source electromagnetism (mCSEM) method, an electromagnetic emitter source emanates a difusive electromagnetic field that traverses water surface and deep into the seabed (CASTILLO-REYES; PUENTE; CELA, 2018) before being captured by an array of receivers. Such receivers can be either placed on the seafloor or also towed along with the source (WILT; ALUMBAUGH, 1998). Figure 3.1 depicts the process using static receivers. These approaches aim to collect the electromagnetic signal readings after they crossed the underlying salt and rock sequences, altering it and, ultimately, imprinting its distinct characteristic onto the signal (CONSTABLE; SRNKA, 2007).

At each moment in time, the electromagnetic emitter is located in a different position underwater, thus defining a collection of virtual electromagnetic sources (one for each emitter's position). When crossing different substances with different resistivity, the wave's amplitude varies. The materials imprint their conductivities, and the receivers capture them, and its information is used as input data to understand the region. Since many receivers capture the same electromagnetic field, it is necessary to establish how multiple

Figure 3.1 – The mCSEM exploration.



Source: Author

receivers' information is combined.

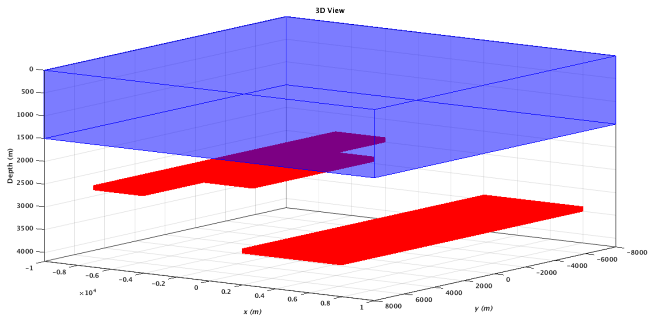
Throughout this data, we can get the model updated with resistivities of the region in analysis through an inverse method, matching the readings to the most probable model of the subsurface substances that generated such readings. This process can use an input to help on the fast numerical convergence. Such model can be composed in two different ways: the first consist of an *a priori* geographic knowledge of the underlying substrate in analysis, and the second can be an initial guess that does not use any *a priori* information. The recovered information is then analyzed to search for known resistivity values of oil and gas.

3.2 The model

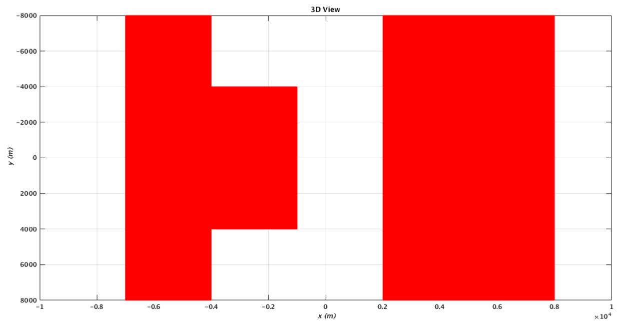
To get back the information collected through the mCSEM process, the application studied in this work uses a representation of real-life as a model. This model has a reasonable approximation of the resistivity in the region in analysis, which benefits the inversion process to converge faster (CORRÊA; RÉGIS, 2017). This model consists of a 3D grid of cells, with its size modeling different regions according to the test case.

A synthetic example is shown by Figure 3.2, in a 3D fashion. The model maps two target regions where specific resistivities are expected (i.e., the regions expected to find petroleum reservoirs). The first target is a T shaped body placed at $1,500m$ below the sea bed; the second is a slab at a depth of $2,500m$ below the sea bed. The model cover a total area of $160km^2$ and also maps regions of water that need to be considered (because they have different resistivities). According to other collected information, each region is set up with a priori resistivity information (i.e., seismic).

Figure 3.2 – 3D synthetic model with 1.5 kilometer of water layer. Target regions are in red.



(a) 3D model view



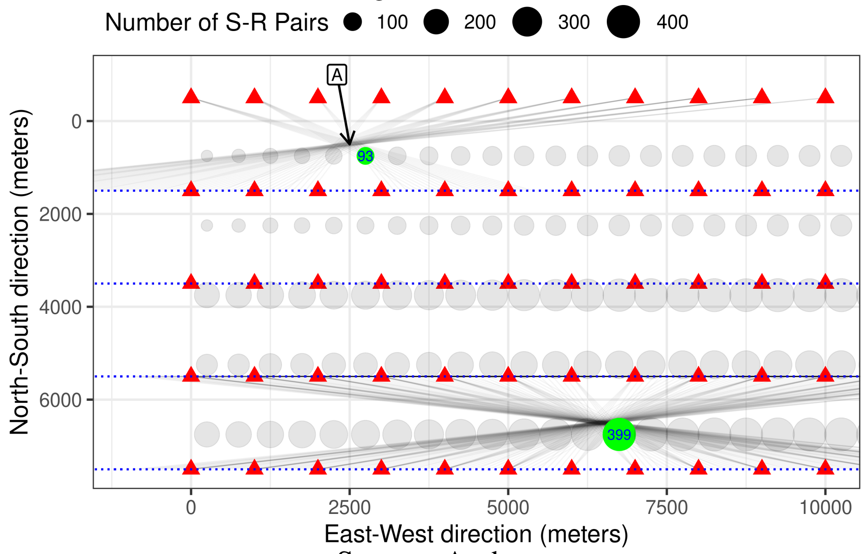
(b) Top view

Source: Corrêa and Régis (2017)

The studied application uses the Common Mid-Point technique (MITTET, 2010) to map the mCSEM data to its model. This is a 1D inversion scheme able to recover 2D structures. In this approach, the data from the 1D inversion is composed of measurements from all source-receivers offsets in the 2D survey that share the same geographical common mid-point. The mapping considers the geographical mid-point between the controlled source and the fixed receivers on the seabed in a determined period of time. It then perform the geographical approximation from all these mid-points with cells of the model, which will be called as the Common Mid-Point cells (CMP cells). This data map contains CMP cell with different quantities of S-R pairs allocated to each and is our virtual model. Such method was improved with analytical derivates and lateral constraints to invert data from 2D surveys by (Silva Crepaldi; Pereira Buonora; FIGUEIREDO, 2011) and is used here to invert data from 3D surveys (CORRÊA; RÉGIS, 2017).

Figure 3.3 depicts a small subset of the resistivity model used in this work, using gray circles with diverse sizes. Each gray circle represents a virtual column in the survey region. The survey data's fixed receivers and the multiple source coordinates are depicted as red triangles (in five horizontal lines) and small blue squares (in four horizontal dashed

Figure 3.3 – Survey data (red triangles are receivers, small blue squares are emitters), combined with the 2D grid of CMPs (circles).



Source: Author

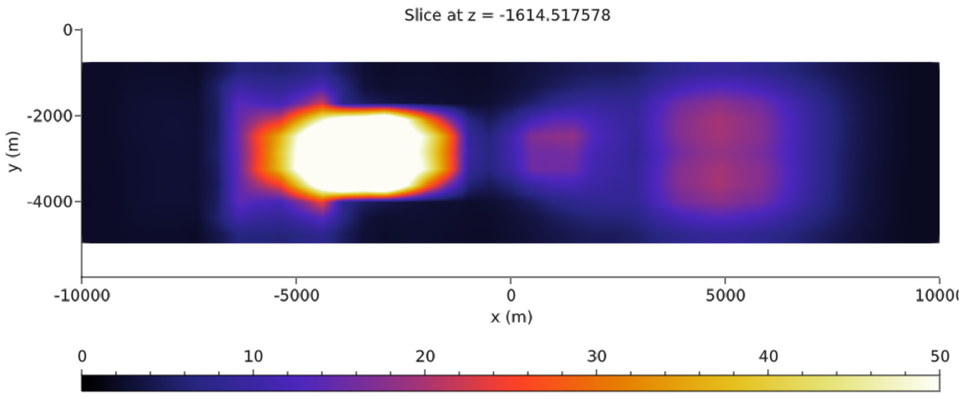
lines), respectively. The data collected by mCSEM can be mapped in multiple CMP cells, where each S-R pair is assigned to different CMPs. Two cells of the 2D grid (circles with numbers 93 and 399) are highlighted in green, representing the diverse quantity of assigned pairs to each CMP cell.

After running the inversion process and recovering data, we have a 3D model to help interpret data or to apply as a first guess in an actual 3D inversion process. Alongside other collected data, such as seismic, geophysics can better map regions where oil and gas can be found. Such an output model will have a more shaped region with specific resistivities (that can be represented from different colors) that indicate where we can find reservoirs in the surveyed region. Figure 3.4 presents an example of a slice of an output model after the inversion process. The inversion successfully delimitates a potential region with high resistivity.

3.3 Inversion Process

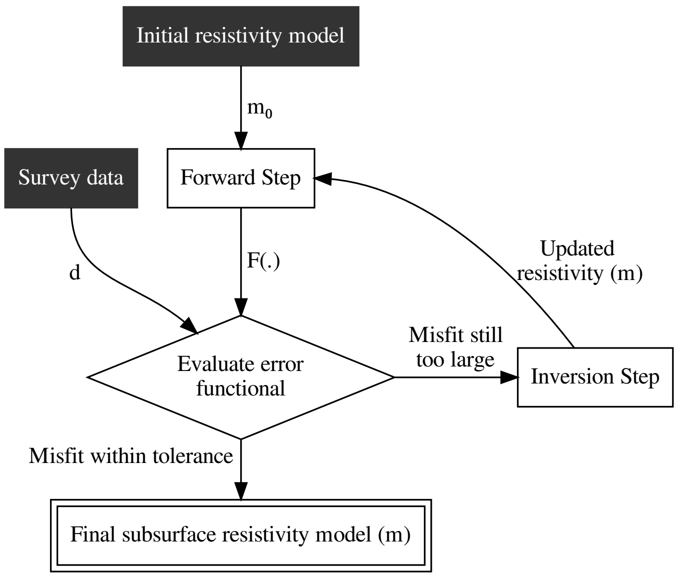
With the model and data defined, we go to the inversion process. Such a process is a collection of different equations that are solved iteratively. This process executes an algorithm based on three phases: forward, linear system solution, and correction (Silva Crepaldi; Pereira Buonora; FIGUEIREDO, 2011). Figure 3.5 provides a general overview of the numerical algorithm, starting from an initial resistivity model m_0 describing the resistivity of each grid point of the 3D inversion volume.

Figure 3.4 – Example of an output model after the inversion process. It is an slice at the depth of 1614 meters collected from iteration 99 of the application.



Source: Author

Figure 3.5 – Data inversion with mCSEM.



Source: Author

In the first phase, we initialize the process by reading the model and the survey data. We then proceed to the mapping of this data to the model using the CMP technique previously described. It generates the heterogeneous distribution, where each CMP cell can have different quantities of S-R pairs allocated to each. Figure 3.3 previously depicted an example of this heterogeneity, with two examples of CMP cells with different sizes.

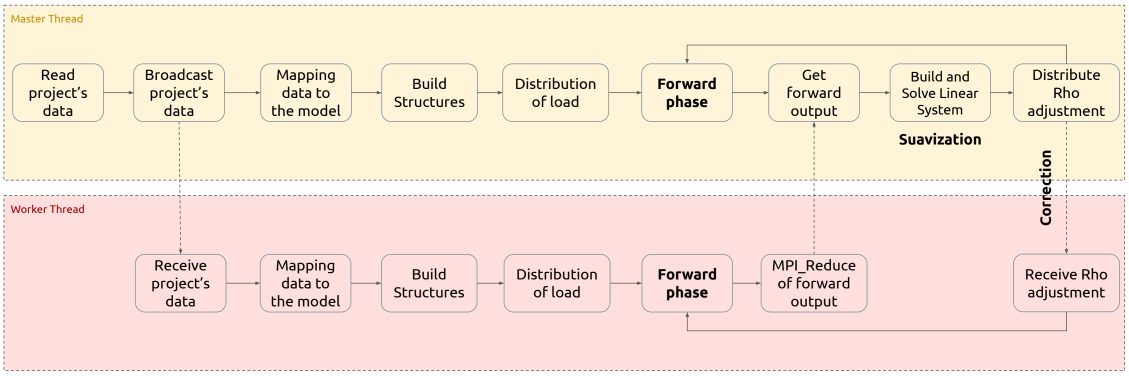
After mapping data, we go to the numerical equations. We have a *Forward* phase where we analytically execute derivatives to calculate the field using Maxwell equations. This is the region of the code that is most computationally expensive. The application also uses the Levenberg-Marquardt (L-M) method (LEVENBERG, 1944; MARQUARDT, 1963; NOCEDAL; WRIGHT, 2006) and it requires solving a linear system at each iteration k, where evaluations of Jacobian functions are applied. This method compares the numerical field with the observed field and finds changes that need to be made, based on the initial measures, to achieve the correct way of the results. On (DAGOSTINI et al., 2021) more details regarding the specifications and the mathematical formalization of such numerical functions are discussed.

The solution of the linear system itself is the next step of the process, where the application run a Cholesky decomposition to solve it. The stopping criteria are the minimum variation of a data misfit and minimum misfit after a specified number of iterations (CORRÊA; RÉGIS, 2017). Finally, a correction phase is executed for the next iteration.

3.4 Parallelization

The application studied in this work performs a 1D inversion process using the CMP domain for 3D data. It uses a coordinator/workers pattern and focuses on parallelizing two phases: the forward and the suavization – in the Cholesky step. Figure 3.6 helps to understand each step of the process, where we have the top of the figure representing the coordinator process and the bottom representing the workers. In the coordinator processs, the application starts by reading all the data collected with the receivers, alongside a configuration file containing execution parameters.

Figure 3.6 – Workflow diagram of the application in study.



Source: Author

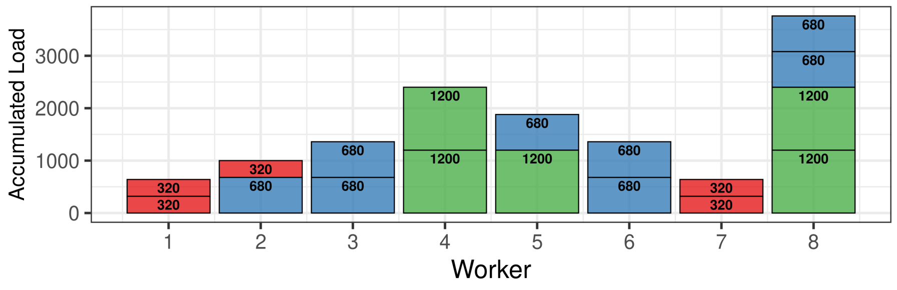
After this reading, the coordinator broadcasts all its data to the workers. Inside each worker (including the coordinator process), the application receive all data broadcasted and executes the mapping according to the model and the parameters provided by the template, using the Common Mid-Point technique. Once this mapping is done, the application prepares the data structures to execute the forward phase and follow to the load distribution. Each worker is responsible for setting their load. i.e., the distribution strategy is executed in each parallel worker.

After finishing the distribution step, the application goes through the forward phase. Now the application will execute the for loop that passes through each CMP cell assigned to the worker and each S-R pair allocated to it. The Maxwell equations are applied to each S-R point at this loop independently. Finishing this computation, all workers apply a `MPI_Reduce` option to deliver the output of the forward phase back to the coordinator process.

With the forward result, we go to the inversion step, where the application builds and solves the linear equations system. This application uses a parallel version of the Cholesky calculation from the SuiteSparse library (DAVIS; HU, 2011) for doing the suavization. After that, the application verifies if it needs another iteration. All this is done uniquely by the coordinator.

The main parallel region of this application is forward. Its parallelization originally applies a hybrid combination of MPI and OpenMP libraries. The load ditribution originally applied in the application consists by the division of the total quantity of CMP cells of the model by the total number of workers. The result of this division is the number of cells that each worker will be responsible to execute. As each of these cells is independent (Silva Crepaldi; Pereira Buonora; FIGUEIREDO, 2011), we can run this phase in parallel without many constraints. Figure 3.7 depicts, with a synthetic load, such distribution strategy. We have 18 “cells” to distribute among eight workers. The X-axis represents the parallel workers, and the Y-axis represents each worker’s accumulated load using the original distribution policy.

Figure 3.7 – A synthetic load characterizing the behavior of the original distribution process.



Source: Author

Then, inside the forward loop, the application will parallelize its process using the OpenMP strategy. As previously explained, the forward loop iterates through each source-receiver pair associated with the cell. This way, each OpenMP thread will deal with the forward math for each S-R pair.

In this work, we focused our attention on the parallelization of the forward step. Due to the way that the application outperforms the distribution of tasks, we find space for improvements in this step related to the distribution of load among the MPI workers. Figure 3.7 help us understand the problem we faced. We can see that, with the original strategy of distribution, the actual load assigned to different workers present significant differences.

4 RELATED WORK

In the last few decades, marine 3D CSEM modeling has been gaining attention inside the Petroleum and Gas market due to the possibility of conducting a sensitive analysis of subsurface resistivity models. Most mCSEM models require a high computational cluster to execute. Therefore, a significant quantity of work has been made to improve the numerical accuracy of inversion applications and achieve a faster convergence to improve its speed. However, few works focus on improving the application's makespan and scalability on a computational science focus, disregarding numerical changes. The next two Sections focus on reported works that aim to improve EM inversion applications. Section 4.1 presents works that apply a combination of algorithms and numerical process changes to achieve better scalability and makespan. In contrast, Section 4.2 present works that only apply computational efforts in existing well-prooved numerical algorithms. Additionally, in Section 4.3, we justify our work's fit scenario and explain its motivation.

4.1 Algorithm and Numerical Improvements Focus

Significant effort was made to focus on different numerical solutions to the CSEM inversion problem. This effort was motivated due to the rapid industrial adoption of this offshore exploration method worldwide (KEY, 2012). The mCSEM method became an essential technique in geophysics exploration (CONSTABLE, 2010). A diverse set of numerical solutions was proposed, decreasing its high computational cost.

There are two main approaches that most of the inversion applications apply to their algorithms: Finite Difference and Finite Elements. Both methods convert Ordinary Differential Equations (ODE) or Partial Differential Equations (PDE) into a system of linear equations that matrix algebra techniques can solve. The Finite Element method minimizes an extremum principle over a family of functions defined by each piece. In contrast, the Finite Difference uses a discrete approximation to integrals and derivates occurring in balance equations (KEY; KRIEG, 1973).

Most works in this topic propose new implementations of such numerical algorithms and changes on the numerical process that brings quality and time reduction to the inversion processes. On (STREICH, 2009), it is implemented a marine CSEM modeling based on a finite-difference frequency-domain (FDFD) approach. Her code relies on the MUMPS direct-matrix solver (AMESTOY et al., 2000) to solve the system of equations using up to 12 MPI processes on a small cluster platform. This solution applies some geographical considerations to use 1D background models to generate 3D outputs. The results present speedup, but the scaling behavior is not good due to memory usage.

The usage of the MUMPS library is not an exclusive characteristic of the previous work. In (WANG; MORTEN; SPITZER, 2018), and (SILVA et al., 2012), the usage of the MUMPS library for solving the linear systems on such applications is reported. The first applies a direct solver with a Gauss-Newton inversion process, using an unstructured mesh to model the seafloor. The second uses Hankel transformations in its inversion process and multilevel partitioning method. In the last, the authors used OpenMP to calculate the primary field. They could achieve a 7.40 speedup using eight threads OpenMP.

Other works implement a 2.5D inversion algorithm using a 3D model based on a direct solver to interpret CSEM data (GRAYVER; STREICH; RITTER, 2013). Its inversion process implements a fully distributed application that relies on the PETSc library (BALAY et al., 2019) for distributed memory linear algebra operations and to access the MUMPS solver to the forward phase. The problem is also parallelized over the primary electric field and frequencies. Even being a more numerical-focused work, this was one of only ones works that take care of load balancing once the workload is statically similarly partitioned to all processes. They do not make explicit which strategy was used for such load distribution. According to reports, the application achieved a performance from $2000s$ in sequential to $\approx 750s$ parallel with ≈ 250 cores, using 64 cores for 4 frequencies of a given study case, presenting a 2.6 speedup.

On (ZHAO; YU; QIN, 2018) work, they formulate their forward problem employing the goal-oriented adaptive finite element method based on the unstructured hexahedral mesh (REN et al., 2013; GRAYVER; BüRG, 2014). They implemented the code using L-BFGS techniques, a numerical quasi-Newton method. They used the library METIS to partition the mesh among the workers. The focus is on improving the math equations to gain performance, not in terms of a computational approach.

A more complete end-user application is presented by (KEY, 2016), which is called MARE2DEM. It is a 2D inversion application that uses a grid of arbitrarily shaped polygons composed of unstructured triangular or quadrilateral grids. The inversion method used in this application also has a forward phase, implemented with an adaptive finite-element method that automatically generates and refines unstructured triangular grids throughout the iterations, ensuring accurate responses as the model conductivity changes. The MARE2DEM is an application of the adaptive mesh refinement (AMR) technique which seeks to minimize the relative error in the EM responses (KEY, 2016). They propose a new Occam inversion approach that they improved numerically and the ScaLAPACK library for linear programming. This work is one of only that presents a load distribution solution that takes care of balancing. The manager processor of the MARE2DEM code keeps a queue with all ready-to-run tasks and assigns the next available task to the next available worker. This dynamic distribution offers a better load balancing to the application once it guarantees that no worker will be idle. This work is a more complete solution from related work found. They present a more robust numerical solution that converges with fewer iterations and takes care of computational characteristics, such as correct parallelization and load distribution.

All of the works mentioned above focus their efforts on combining numerical process changes to achieve a better makespan. Most use simple models to execute the inversion process, scoped to grid and cells models. On the other hand, some works apply unstructured meshes to achieve better numerical results. However, only one work brings a discrete discussion regarding load balancing or the study of computational concepts that could benefit their works. Even discussing it, its primary goal is to discuss and present the numerical improvements applied to the application. It is the main characteristic of all related works presented in this section.

4.2 Computational Improvements Focus

From another point of view, some works aim to improve the time spent in the mCSEM inversion process taking care of computational improvements. Most of them already have verified numerical methods but cannot correctly use the computational resources or extend them to new devices and accelerators.

Two recent works relate the parallelization of the forward step of the inversion process into GPUs. In (SOMMER et al., 2013), they present an implementation of a 3D marine CSEM modeling on GPUs, focusing on the parallelization of the forward step from a finite difference method. The original application was executed on one standard CPU kernel and could take up to 1 hour to run complex models. Their approach achieves significant speedups using standard libraries such as CUBLAS and CULA, where the latter provides an optimized GPU QR algorithm. They could achieve a speedup of 9 to SLDM and 35+ in Eigensolver in GPU. CULA-library presented an improvement from the parallelized CPU version of the LAPACK eigensolver by a factor of 10.

The second GPU-related work is (YANG, 2021). Using the Finite Differences Time Domain method, they implemented this new solver with CUDA, designing the application to precompute every detail before porting the code to the GPU step. They used a shared memory with a tiled algorithm to improve the usage of the L1 cache. Their solution did not scale well using OpenMP. They used a GTX860M to conduct experiments. Using a 1D model data, they compared their work with (KEY, 2009) and against a version of their application implemented with CPU/OpenMP. They achieved a makespan from 723s on a single thread to 18.1s on GPU, representing a $40\times$ speedup.

Some works rely on applying a mix of computational efforts to improve the application's performance. They use existing and well-known parallelized libraries and tools and change other parts of the algorithms to run in parallel. This is the case of (LONG et al., 2020) and (da Piedade et al., 2021). Both works used different versions of the Pardiso library (ALAPPAT et al., 2020), a parallel direct solver to perform linear system executions. The second work compared the behavior of two finite methods, one using Nodal and the other using Vector FEs. With deeper analysis, they concluded that the Vector solution brings the advantage of avoiding difficulties when there are discontinuities in the EM field found with nodal.

A more similar solution to the proposed contributions from this work was done by (PETHICK; HARRIS, 2016). They developed a Java software that did not interfere with source mCSEM codes. The application englobe an mCSEM algorithm and distribute the load among the workers in execution that can also scale in a grid of computers. The key point of their work is that it considers inversion applications only composed by forward and inversion steps, without any other step further. They also applied this macro-parallelization with other electromagnetic surveys similar to mCSEM. They focus their efforts on distribution strategies implemented in a macro-parallelization method.

4.3 Discussion

The previously presented works bring us the idea from two perspectives: mixing improvements on numerical methods to achieve performance and focusing on computational efforts. The first scenario presents works that their typical strategy relies on standard parallel linear algebra solvers like MUMPS. Meanwhile, no one presents a discussion about the impact of the workload partition on parallelization.

Our work fits the second scenario, where we aim to apply load balancing techniques to improve application performance. The present work focuses on computational science improvements, disregarding numerical changes in the inversion process. We could achieve significant results by applying minor changes to an existing code and partitioning the load among homogeneous and heterogeneous workers, alongside applying correct heuristics to do such distributions.

So far, no work focused on applying load balancing techniques exclusively to an mCSEM application was found. The closer work presented was the one done by (PETHICK; HARRIS, 2016), discussed in Section 4.2. However, they did not pay specific attention to this distribution's possible load imbalance issues or report any specific study related to load balance solutions once they considered it as an *embarrassingly parallel* problem. Our work identifies severe load imbalance issues in the studied application, resulting from the problem domain's intrinsic unbalanced partition. Our contributions aim to show how well-known load balancing techniques from literature can impact this real-life application.

5 CONTRIBUTION: FINDING THE IMBALANCE

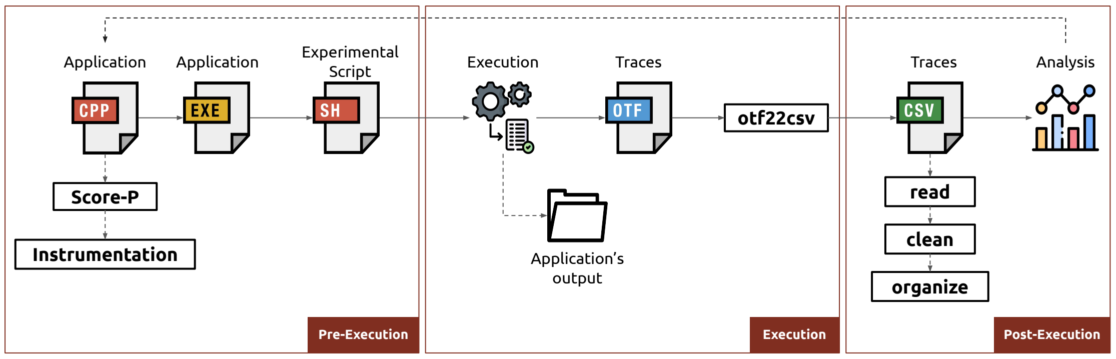
Our first contribution presents the origin of the application's load imbalance. To find this lack of performance, we apply a performance evaluation with code instrumentation and statistical analysis. We generated traces that allow us to precisely define what composes the load of the application and the reason for the load imbalance. The results presented here are the baseline for the following two contributions, that are the chosen load balance heuristics described on Chapter 6, and the capacity planning proposal on Chapter 7, due to these definitions.

We present how we perform this investigation in Section 5.1. Next, we discuss how the mapping method impacts applications execution in Section 5.2. We then show how we identify the problem in Section 5.3. We finish this chapter presenting final remarks regarding this contribution on Section 5.4.

5.1 Methodology

To execute our investigation, we applied the following methodology shown by Figure 5.1. We instrument the application's code to capture traces of the application's execution and implemented user regions instrumentations to get the duration of each inversion step described in Chapter 3. We use Score-P (KNüPFER et al., 2012), an open-source library for code instrumentation in C, C++, Fortran, and Python languages. Our goal is to find the origins of the application's bottlenecks.

Figure 5.1 – Methodology steps executed in this contribution.



Source: Author

With the code regions identified, we create an experimental script to execute. We use machines present in the *Parque Computational de Alto Desempenho*, PCAD, from the

Institute of Informatics at UFRGS. We utilize four nodes of the **Draco** cluster, composed of two Intel Xeon E5-2640 v2 2.00GHz per node, with eight cores per processor. Each node has 64GB DDR3 RAM, connected by a 1Gbit/s network. All nodes run the Debian 10 operating system, with Linux kernel version 4.19.0-10-amd64. The application was compiled using OpenMPI 3.1.4 and GCC 8.3.0 with the following compiler flags -c -O3 -g -funroll-loops -static.

As input data, we exploit one synthetic study case, which consists of a resistivity model with 41×11×89 (width, length, depth) cells, for a total of 451 CMPs, approximating a volume of 20×7.5×4.2 cubic kilometers. The survey data consists of 1,061,046 data points from 189 fixed receivers and 176,841 controlled-source positions and contains values associated with three frequencies for each S-R pair.

After running the application, we collect OTF2 traces generated by Score-P and convert them to CSV using the `otf22csv` converter (SCHNORR, 2019). We then read and sanitize the data, generating output data ready for analysis. We use a science workflow implemented with the R language (R Core Team, 2020) with the tidyverse (WICKHAM et al., 2019), and ggplot2 (WICKHAM, 2016) packages to analyze the results.

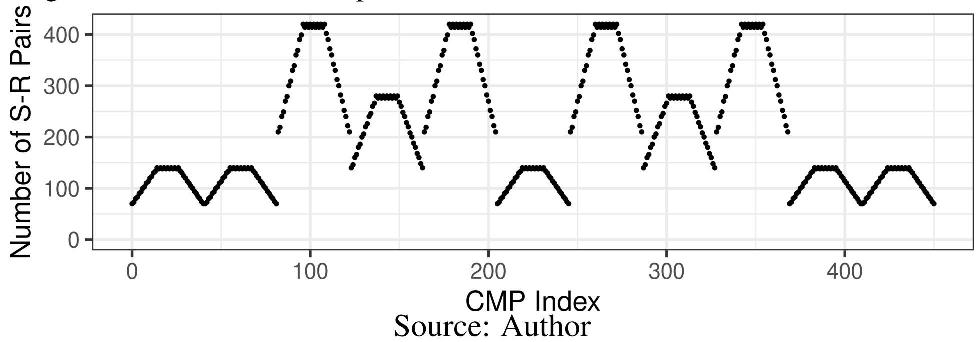
5.2 Mapping Data to the Model

As previously stated, the studied application executes a nonlinear function with the electromagnetic field in an independent fashion for each CMP cell. To be considered in some aggregation to be allocated in a CMP cell, each geographical common mid-point of each S-R pair must match a cell in a resistivity model. With this strategy, we have the mapping setup as previously depicted by Figure 3.3.

The application executes such a combination in the initialization phase. For each S-R pair, the application computes the mid-point and then finds the closest cell of the resistivity model by relying on their geographical coordinates. This combination of the application parameters and the survey data results in an imbalanced number of S-R pairs allocated for each model cell.

To better understand this unbalancing matching process, we instrumented the code to gather information regarding the organization of the cells. We generate outputs that extract the quantity of S-R pairs allocated to each CMP cell for the studied model. Figure 5.2 depicts this unbalancing, with the number of pairs on the Y-axis associated with each cell of the resistivity model in the X-axis. We can see a four-time difference between cells with more pairs allocated to those with fewer in this model.

Figure 5.2 – Number of S-R pairs associated for each CMP of the studied model.

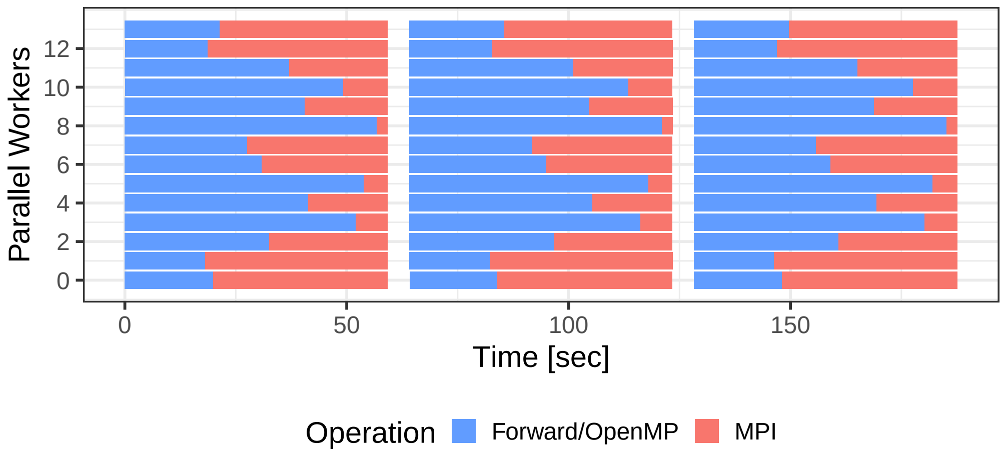


Source: Author

5.3 Problem Identification

The temporal traces gathered from the application introduced the load imbalance among machines. Figure 5.3 shows the results, where the X-axis is the time in seconds, and the Y-axis represents the parallel workers involved in this execution. We run the application with 14 MPI workers with 8 OpenMP threads per worker, having 112 parallel threads. The colors of the plot represent the different operations being executed in each second of execution. We can see that the regions identified as OpenMP operations take different durations to execute throughout the workers. This difference is what we consider as load imbalance.

Figure 5.3 – Temporal data collected in one execution with three iterations. Colors represent different operations detected during execution.

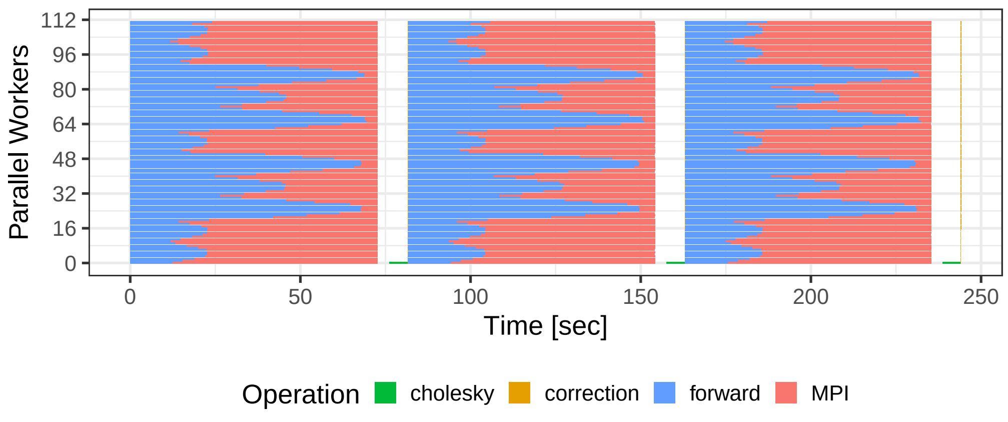


Source: Author

Investigating the code deeper, we noticed that the OpenMP region corresponds to the forward phase of the inversion algorithm. We also observed that it is a loop, and it iterates among the S-R pairs of each CMP cell assigned to the worker. We then identify the other phases in code, taking the numerical inversion process as the baseline, previously described in Chapter 3. We could correctly identify where the application executes the solution of the linear system and the correction after that.

For the next execution, we disabled OpenMP threads and ran the application with 112 MPI workers, capturing the regions of the code based on the algorithm steps. Our goal here was to check if the load imbalance observed in Figure 5.3 comes from the forward phase indeed. Figure 5.4 depicts the observed behavior. The X-axis still represents the time in seconds, while Y-axis represents the parallel workers involved in the execution. The colors in this plot represent the three main regions of our application: forward, smoothing/Cholesky, and correction, alongside communication regions identified as MPI.

Figure 5.4 – Execution behavior of the application executing only with MPI, presenting the timespace of the three main regions of the inversion algorithm.

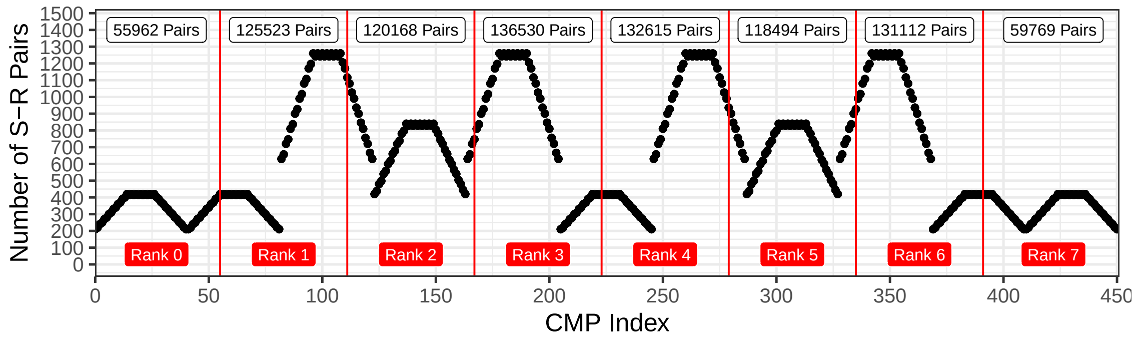


Source: Author

This new visualization made it more evident that the imbalance comes from the forward phase (blue region) and the different time duration among the parallel workers. Such difference, as previously explained, indicates the load imbalance of the application. As also previously discussed, we know that the application divides the total quantity of CMP cells in the study case among the number of MPI workers in the execution. Moreover, the main loop of the forward region iterates throughout all the S-R pairs aggregated to each CMP cell allocated to the worker. This way, the actual load of the forward loop is given by the quantity of S-R pairs aggregated into a cell, not by the cells themselves. We can also conclude that the application's load is static due that each iteration presents the same distribution and similar duration.

Figure 5.5 helps to identify the unbalanced distribution among the workers. It depicts all 451 CMPs in the X-axis and the number of S-R pairs in the Y-axis. The red lines represent eight different MPI workers and the region of load that each one of them will be responsible for. The red labels indicate the MPI rank/worker, and the white labels how many S-R pairs this rank has to compute. We can observe that Rank zero, for example, will have considerably less work than Rank one because the first cells of the model have fewer S-R pairs aggregated to it than those that will be assigned to Rank one to compute.

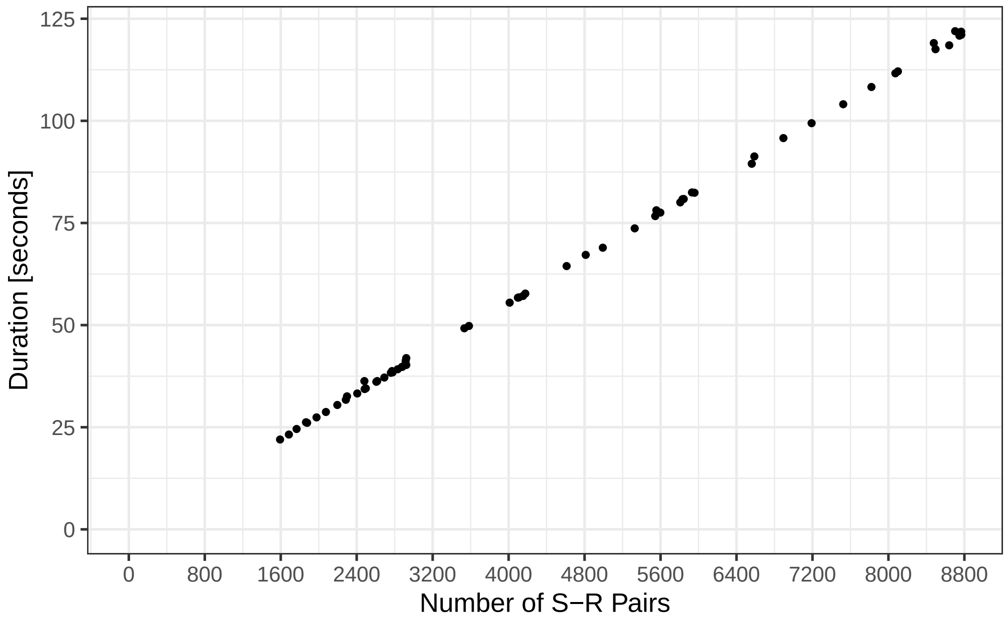
Figure 5.5 – Behavior of the original distribution – it divide the cells among workers, disregarding its own heterogeneous weight.



Source: Author

We confirm that the number of S-R pairs is directly related to the time spent by the application. We update our instrumentation to also collect the duration of each CMP cell to execute. We ran it several times and concluded that the execution time has low variability across iterations. Figure 5.6 shows the direct relationship between the processing time (Y-axis) and the quantity of S-R pairs (X-axis). The numbers presented in

Figure 5.6 – The number of S-R combinations explains processing time.



Source: Author

X-axis represent the total amount of S-R pairs that different parallel workers received to compute. We can see that, as the number of S-R pairs increases, the duration increases linearly. This way, the worker who received 8800 pairs to compute will take double the time to execute the forward phase than the worker who received 4400 pairs. We apply ANOVA (ANalysis Of VAriance) (GEORGE et al., 2005) to confirm the direct association between the number of combinations allocated among MPI ranks and their execution time. The ANOVA results confirm our hypothesis, with a statistical confidence of 99.7%.

5.4 Final Remarks

With all this data, we accurately identified the application's load imbalance. The original strategy to distribute the load among workers disregards the actual load of the application, which are the S-R pairs aggregated to each CMP cell. This load is static throughout the iterations and presents a linear behavior with the duration of the application's execution. These characteristics will be relevant for further contributions once they allow us to search for complete solutions that can be applied to this kind of problem. Finding this imbalance – and going deeper into its characteristics and the origin of the problem – make all the rest of this work possible.

We aim to apply a solution that minimally interferes with the existing code and strategy to solve this load imbalance. For this reason, we do not want to split the CMP cells to parallelize the process, as some other works do (PETHICK; HARRIS, 2016). Such conditions led us to a number-partitioning problem with existing solutions to apply. Our next goal is to apply and study the application's behavior with heuristic solutions, as we will see in the next Chapter.

6 CONTRIBUTION: APPLYING HEURISTICS TO THE PROBLEM

From the problem identification in Chapter 5, we seek to find solutions. Our goal is to deliver an improved real-world application by applying existing and well-known load balancing strategies in the literature. Such an approach match with the main objectives of the applied computational field, where we aim to increase computational solutions in real-world scenarios.

We found two solutions well suited for homogeneous resources: Round Robin and Sorted-Greedy. These two solutions are referenced and well studied to solve scheduling problems with static load. The last one correlates to number-partitioning problems, the category of the problem we faced in this work. As previously discussed in Chapter 2, Sorted-Greedy is the best solution for partitioning problems with one multi-way variable.

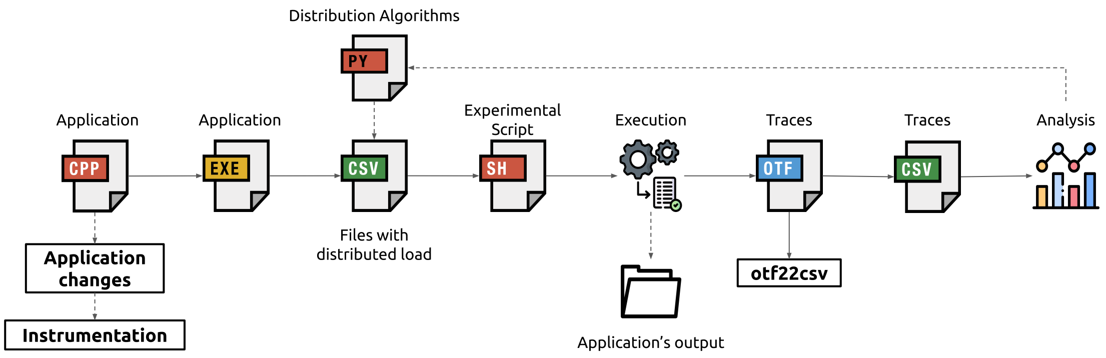
Furthermore, we aim to explore other possible solutions as we were not limited to homogeneous environments. We seek to investigate schedulers for heterogeneous scenarios and propose the application of Area Bound Estimation Guided (ABE-Guided) and the Heterogeneous Earliest Finish Time (HEFT). The first algorithm is an adaption from the original ABE linear program, once we use the output that this existing method already provides adapted to our needs. For the last, HEFT fits our problem once we have the challenge to exactly achieve what the algorithm's name proposes.

This chapter aims to present the results and analyze our best solutions for homogeneous and heterogeneous computing resources. Section 6.1 starts presenting the materials and methods applied to this contribution and how we executed our experiments. We then follow to Section 6.2 that brings a theoretical analysis of the behavior of homogeneous algorithms and some important insights this analysis led us to have. Sections 6.3 and 6.4 focus on presenting and discussing results obtained with real executions of all four schedulers. We close this Chapter presenting that our schedulers did not affect our application's numerical accuracy on Section 4.1.

6.1 Materials and Methods

For this contribution, we apply the following methodology depicted in Figure 6.1. According to the already described method in Chapter 5, we gather data from instrumentation. We code and execute our load-balancing solutions from this data out of the application. We use this methodology due to the benefit of doing minor changes to the original code to test all the different proposed algorithms. We add two lines responsible for reading the files with the distribution previously made on the application's code.

Figure 6.1 – Methodology applied to this contribution. We used the intrumentation made previously to analyze the results provided from our distribution algorithms.



Source: Author

The scheduler algorithms were initially coded in Python and R, receiving as input the mapped data to the model gathered in a previous execution of the application for each model. This previous instrumented execution resulted in three collected information: the id of each CMP cell, the quantity of S-R pairs allocated to each cell, and the time spent on each execution. All this data was cleaned and used as the input for the algorithms for distribution. The output of these solutions was two files containing the total load assigned to each parallel worker (in the quantity of CMP cells) and the mapping for CMP and worker to execute. The application, in its execution, read these two files. After running validations to the efficiency of the methods, we implement them in C language as new modules for the application.

We use the synthetic study case already presented in Chapter 5 to test the heuristics. To execute our homogeneous and heterogeneous solutions, we use four different groups of machines – which we called here as partitions – from the *Parque Computacional de Alto Desempenho* (PCAD). Table 6.1 summarizes its characteristics. We have a total of 15 machines and 304 physical cores. They run the Debian 10 operating system with Linux kernel version 4.19.0-18-amd64. We compile the application using OpenMPI 3.1.4 and GCC 8.3.0 with the following compiler flags -c -O3 -g -funroll-loops -static.

We execute the application with our study case and follow the practical evaluation framework used in the imbalance investigation to have comparable data. For this reason, we analyze the data with R and ggplot. We were also able to do simulations with data using the R language to better understand it, as we shall see in what follows.

7x Draco	
Architecture	x86_64
Model	Xeon E5-2640 v2
Frequency	2.0GHz
Cores Per Socket	16
Threads Per Core	32
Socket(s)	2
RAM	64GB(1-6) DDR3 128GB(7) DDR3

(a)

5x Hype	
Architecture	x86_64
Model	Xeon E5-2640 v3
Frequency	2.3GHz
Cores Per Socket	10
Threads Per Core	20
Socket(s)	2
RAM	128GB DDR3

(b)

2x Cei	
Architecture	x86_64
Model	Xeon Silver 4116
Frequency	2.1GHz
Cores Per Socket	12
Threads Per Core	24
Socket(s)	2
RAM	96GB DDR4

(c)

1x Blaise	
Architecture	x86_64
Model	Xeon E5-2699 v4
Frequency	2.2GHz
Cores Per Socket	22
Threads Per Core	44
Socket(s)	2
RAM	256GB DDR4

(d)

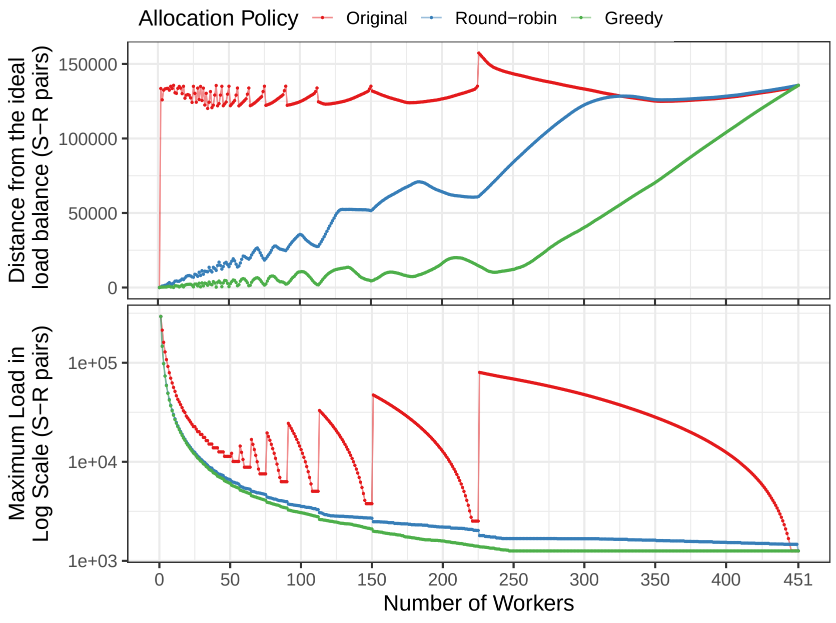
Table 6.1 – Specification of machines used in the executions.

6.2 Theoretical Analysis on a Homogeneous Environment

As the application's load is linear and static, we perform a theoretical analysis on an homogeneous environment. We implement computational simulations for the homogeneous scheduling policies (Round Robin and Sorted-Greedy) before executing them on the real scenario. Our goal is to compare their behavior and theoretical gains against the Original distribution. We carry out a strong scalability analysis by computing, for each policy, the total distance from the ideal load balance and the maximum load obtained for a given number of workers. The analysis presented a good starting point using a representative workload and was essential for us to learn what to expect on the real executions.

Figure 6.2 (top) depicts the total distance from the ideal load balance (on the Y-axis) for each approach (colors) as a function of the number of workers (X-axis), from 1 to 451 (the number of CMP cells in the resistivity model). We simulate the CMP load distribution to verify which approaches have better behavior by achieving the lowest variability in the workload distribution. Round Robin improves the scenario against the Original heuristic, and Sorted-Greedy improves against both previous heuristics. The three algorithms present the same maximum load with a single worker since only one can process all CMPs. Meanwhile, when the number of workers equals the number of CMP cells, the maximum load equals the largest CMP cell size. Sorted-Greedy delivers the best solution allocating the S-R pairs taking into account the least-loaded worker to do the distribution, which explains the smoother reduction in the maximum load. When the number of workers exceeds half the number of CMP cells, all three approaches increase the total distance from ideal.

Figure 6.2 – Theoretical distribution of load with 1 up to the number of CMP cells of the study case, presenting the distance from ideal mean from these settings (top) and the maximum load allocated to some worker (bottom).



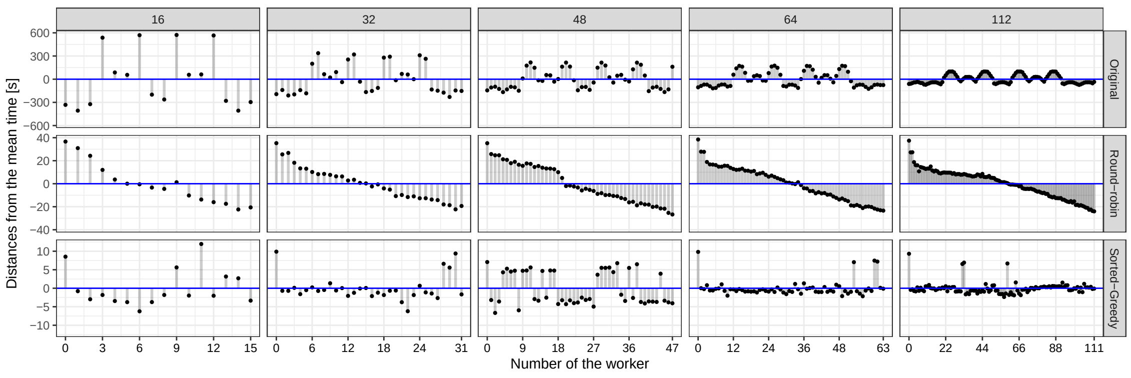
Source: Author

This analysis gives us an insight into the application's behavior according to the input data in execution. Due to its static characteristics, knowing the time spent in each executable resource allows us to accurately predict the application's behavior and how much time each iteration of the inversion process will cost. This conclusion was a valuable insight for the capacity planning that we will focus on in Chapter 7. We started the real executions with this initial idea of what to expect from the selected algorithms for homogeneous environments.

6.3 Heuristics in Homogeneous Environments

We first observe the distance from the ideal load balance, in seconds, running three iterations of the code from 16 up to 112 physical cores with one worker per core. Figure 6.3 depicts, along the Y-axis, the distance (points, in seconds) from the ideal load balance (the blue line) for each worker (X-axis). We compute the ideal load balance by dividing the total time spent on computation for those three iterations divided by the number of workers of that experiment. We depict the results using facets: the cases with 16, 32, 48, 64, and 112 workers are the columns, while rows represent the three schedulers (Original, Round Robin, and Sorted-Greedy) from top to bottom, each one with their Y-scales to reflect the deviation from the mean. We observe in Figure 6.3 similar results to the theoretical analysis made previously, where Sorted-Greedy and Round Robin solutions present smaller distances from the ideal mean.

Figure 6.3 – The distance from ideal load balance for every worker considering five total number of workers (columns) and the CMP scheduling policies (rows).



Source: Author

The Original scheduling policy (top row of Figure 6.3) presents the most diverse distribution, having considerable differences from the ideal time, especially with 16 workers. As the number of workers increases, the distances diminish but follow the same ups-and-down pattern due to combination of S-R pairs from the input data to the resistivity model. The Round Robin scheduler (middle row) attributes higher load for the first workers, presenting a positive distance from the ideal time when compared with the last workers. We expect such behavior because the algorithm distributes the computation of CMPs in ascending worker's number order from the most to the least costly CMP. Finally, the Sorted-Greedy policy (bottom row) presents the best fit, showing the smallest distances from the ideal load balance and a more even distribution.

The most extensive scenario with 112 workers in Figure 6.3 has some workers far from the ideal distribution in the Sorted-Greedy policy. That is the case of worker 0, for instance. These discrepancies in the load imbalance appear because of CMPs' different computational load. An alternative design of splitting a CMP's computing across many workers would probably make a perfect load balance, but that would require more profound changes in the application's code. With minor changes, we observe that both

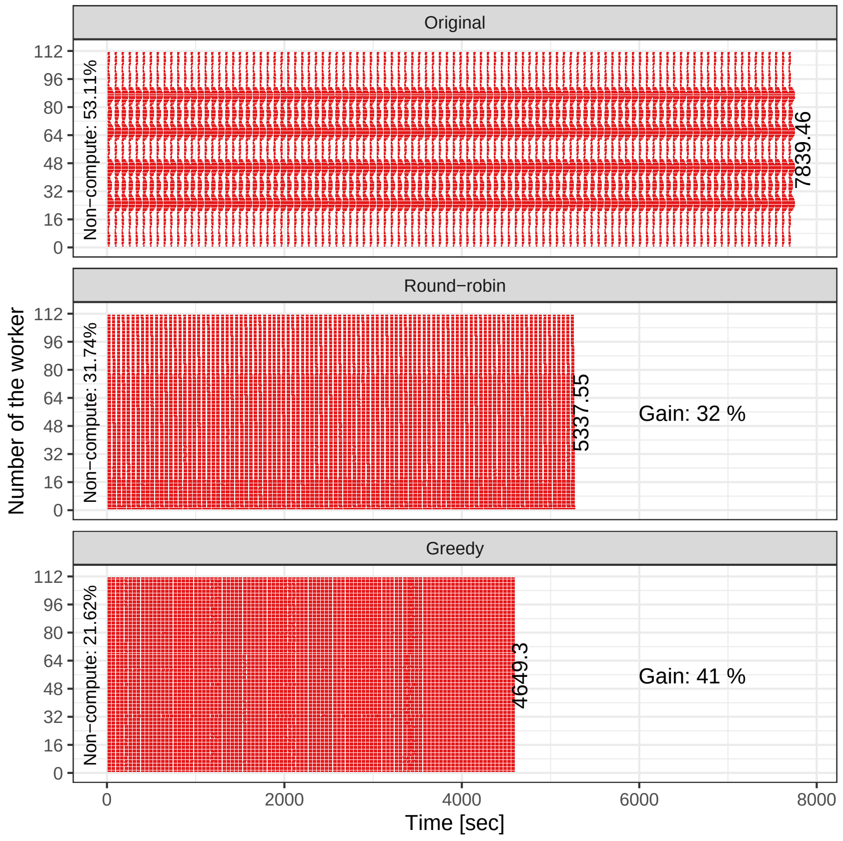
Round Robin and Sorted-Greedy policies improve dramatically against the Original one, presenting a reduction of the distance time from more than 100 seconds with the Original strategy to less than five seconds with Sorted-Greedy in the 112 workers scenario.

Our next goal was to check makespan reduction with the new schedulers. We changed the application to run with specific pre-defined quantity of iterations (even if we already achieved the goal-misfit) to get improved data. We executed the application with the three different distribution algorithms using 112 workers (the maximum number of physical cores available for this experiment with a unique partition). Using Score-P and our manual instrumentation of the application, we collected trace data to track down the most costly operation (the forward step). All other parts of the application code have a non-significant time footprint to appear in the plot with such scale. Figure 6.4 presents a space/time view with colored rectangles to represent the forward step for every worker (on the Y-axis) during 100 data inversion iterations along the execution time (X-axis). We facilitate the comparison among policies (facet rows) by aligning the time on the X-axis. We augment the view with the makespan (the rightmost number) and the percentage reduction for the Round-robin and the Sorted-Greedy policies against Original (marked as Gain in the rightmost white area).

The core result is the reduction in execution time of 32% for the Round Robin, and of 41% for the Sorted-Greedy policy, when compared to the Original policy. Reduction is from ≈7761s in the Original policy to ≈4603s with Sorted-Greedy. Another observation is the amount of time spent doing other non-compute operations. We show, on the leftmost part of each facet in Figure 6.4, the average of non-compute time considering all workers. With the Original policy distribution, most of the execution time is spent on non-computing operations (53%). As expected, the ratio considerably decreases with the Round Robin (31%) and the Sorted-Greedy policies (21%) because of better load balance. This better use of resources implies on the Sorted-Greedy's best performance among the studied policies. While results are impressive, there is still room for improvement in the Sorted-Greedy scheduling policy.

Moreover, we seek to investigate the speedup of the application comparing schedulers. We compare the two proposed heuristics against the Original using 1, 2, 4, 8, 16, 32, 64, and 112 MPI processes with the Original model. We compute the mean speedup

Figure 6.4 – Space/time view of the forward step of 100 iterations for the Original, Round-robin, and Greedy allocation policies.



Source: Author

and its variance applying the following equations:

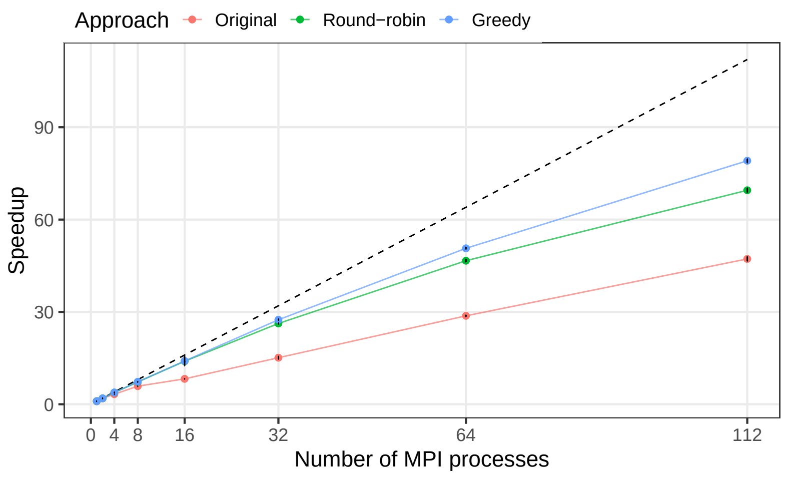
$$\mathbf{E}(\text{speedup}) = \mathbf{E}(\text{Original 1 proc}) \times \mathbf{E}(\frac{1}{\text{new\_distribution } n \text{ proc}}); \text{ and}$$

$$[!htb]\mathbf{Var}(\text{speedup}) = \mathbf{E}(\text{Original 1 proc}^2) \times \mathbf{E}(\frac{1}{\text{new\_distribution } n \text{ proc}^2}) - \mathbf{E^2}(\text{original 1 proc}) \times \mathbf{E^2}(\frac{1}{\text{new\_distribution } n \text{ proc}}).$$

where $\mathbf{E}(speedup)$ denotes the sample mean of the *speedup*. We multiply the speedup of our new distribution heuristics by the sample mean of the Original to achieve a comparable speedup with it. Our speedup goal here is present the gains obtained with our new distributions against the original one.

Figure 6.5 presents the speedup curves for each approach with error bars representing 99.7% confidence interval. Both Round Robin (green) and Sorted-Greedy (blue) clearly outperform the Original (red) approach with any number of MPI processes. Setups using from 1 to 16 processes can execute on a single node since each node has 16 cores. We can first observe that, while the Original approach already presents scalability issues in this single-node scenario, the two new strategies perform closer to the ideal speedup.

Figure 6.5 – Speedup of 1 up to 112 MPI processes. We run three iterations of the application. The baseline is Original with one process. The dashed black line represents the ideal speedup.



Source: Author

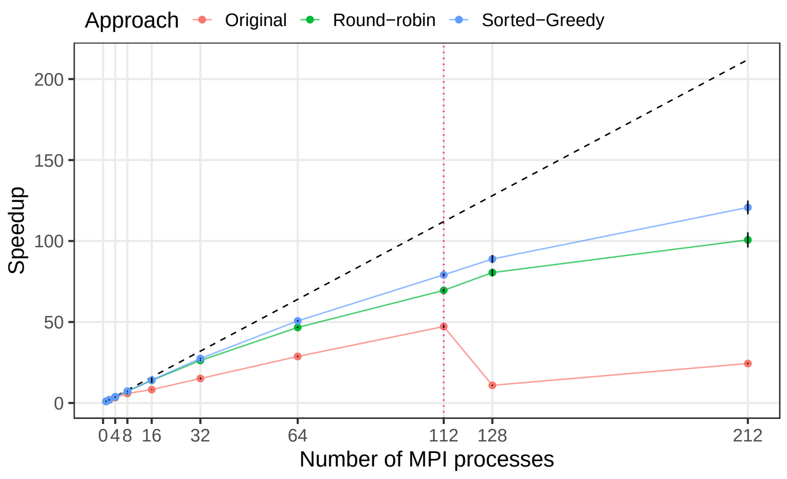
The speedup increase from both new scheduling policies became more evident as the number of workers increased. With 112 processes, i.e., using all available Draco nodes, we observe speedups of ≈80× for Sorted-Greedy, ≈70× for Round Robin, and ≈47× for Original. With this setup, the Round-robin approach reduces the execution time compared with the Original distribution by 32% to ≈163s, while Sorted-Greedy decreases it by 40.3% to ≈143s. These results can prove that such well-known algorithms from the literature proved adequate for this real-world problem.

6.4 Heuristics in Heterogeneous Environments

We seek to compare the theoretical strong scaling study with real executions. For this, we need to run the speedup executions in another partition from PCAD once the Draco cluster does not have more than 112 MPI workers. We then run the application in both Draco and Hype partitions. Once our measurements have indicated that the processor frequency explains 97.4% of the compute capability difference between the two partitions, we change the frequency of Hype processors to be equal to Dracos', reducing it to 2.0Ghz, to have more accurate results. For each combination of scheduling policy and MPI processes. This way, we could scale the speedup to use 128 and 212 MPI processes. Figure 6.6 depicts new results. The results presented before the vertical dotted pink line represent executions made only at the Draco partition. After that, we add the Hype partition to extend the scalability test.

In Figure 6.6, we can observe that our theoretical analysis for maximum load for

Figure 6.6 – Speedup of many MPI processes from 1 up to 212 using two different partitions with the same processor's frequency. We also run the application with three iterations, and the baseline is Original with one process. Dashed black line represents the ideal speedup.



Source: Author

the Original policy is correct in real executions. When the number of workers comes closer to 120, the Original behavior decreases significantly, presenting a speedup similar to when using eight workers. As explained previously, the Original distribution can present such unbalanced and irregular behavior as the number of workers changes because it neglects the load linked to each CMP. The performance of Round Robin and Sorted-Greedy are very similar when using from 1 up to 32 processes (2 nodes). The gap between them becomes clearer when using 64 processes (4 nodes) and increases with 212 processes (12 nodes, using two partitions). This gap is somehow expected since the distribution implemented with the Sorted-Greedy strategy is more concentrated around the mean, favoring scalability.

When we increase the number of MPI processes to 128 (now using all Dracos plus one node of Hype partition), we still see improvements with Round Robin and Sorted-Greedy, with speedups of ≈81× and ≈89×, respectively. The highest values are presented with 212 MPI processes for both Sorted-Greedy (≈120.7×) and Round-robin (≈100.7×), while the Original approach has substantially deteriorated from 128 or more processes. In other words, both new distribution policies enable application scaling with a higher number of workers, which is not possible with the Original code. Even if the observed speedup values are getting farther from the ideal speedup, this is common in a strong scalability test. Increasing the number of processes reduces the workload processed by each one, then highlights the impact of communications and synchronization steps. Despite that, we can see that these two new schedulers significantly improve the application scalability.

All these observations made with these real strong scaling experiments motivated us to extend our solution for heterogeneous environments, considering real heterogeneous settings. Our main goal is to achieve load balancing using machines with different compute capacities without changing its processor's frequency, exploiting the system-level heterogeneity to our problem.

Heterogeneous environments are gaining attention in the scientific community, mainly on the intra-node approach. Mixing computations on different hardware allow applications to achieve better time performances. Due to the heterogeneous characteristics of our local cluster, which is composed of a large set of different machines, we saw an opportunity to explore further improvements to our application. This solution can also validate the theoretical analysis with more workers (to check the behavior when we have more workers than half of the total quantity of CMP cells to compute).

We seek to find new load balancing heuristics that concern different machines' capabilities alongside the heterogeneous characteristics of the load. We found two possible solutions to our problem, all depending on previous benchmark data in each involved resource: a solution based on the results from ABE linear programming and another from HEFT. We adapt the HEFT algorithm to predict the minimum makespan with the maximum usage of the resources disregarding communication times and task dependencies. The simplified HEFT algorithm predicts the duration of each task inside each worker, and it will attribute the task to the worker where it presents the best total makespan.

For the ABE-guided solution, we change the input of the ABE linear program in the way that the tasks T represent our group of cells C, where the quantity of S-R pairs linked to them determine these groups. For example, using as a base the CMP distribution of Figure 5.2, we have a group of cells with 1200 pairs, others with 460 pairs, and others. We have updated the formal definition of ABE to the following equations:

$$\text{minimize:} \qquad makespan \tag{6.1}$$

subject to:

$$\forall c \in C: \qquad \sum_r n_{c,r} = N_c \tag{6.2}$$

$$\forall r \in R: \qquad \sum_c n_{c,r} * w_{c,r} \leq makespan * N_r \tag{6.3}$$

$$\forall c \in C, r \in R: \qquad n_{c,r} \geq 0 \tag{6.4}$$

Where we changed the t tasks for c determined group of cells C with the same quantity of S-R pairs linked to it, r is the type of resource R (that are the different machines where we are executing the code), and $w_{c,r}$ is the meantime which the determined group of cells c takes to execute in a determined resource. We implement our ABE linear program to get as output the final makespan and the array with the distribution set that the linear program uses to achieve this mark. Once that such distribution set only determines the number of cells of each group to each resource, we then use this distribution array as input for a Sorted-Greedy phase, which will allocate each specific task to each resource.

We follow the same test approach presented with homogeneous solutions to validate our new heuristics. We first executed the three new heterogeneous approaches and compared them to the behavior of our best homogeneous solutions (Sorted-Greedy) running in a heterogeneous environment. We selected three different machine combinations scenarios to execute this test. They are summarized by Table 6.2. The first use all of our available resources, using all available machines from our five partitions **Draco**, **Hype**, **Cei,** and **Blaise**, giving a total of 304 workers (A). The second uses three partitions and 112 workers (B), and the third also combines three partitions and 112 workers but uses a different combination (C). Scenario A aims to observe the behavior of each algorithm when using more than half of the CMP quantity of the model in study. The B and C scenarios aim to compare our homogeneous scenario (composed of 112 workers). We want to demonstrate possible improvements that these heterogeneous solutions can bring to the application even using the same quantity of the maximum available number of homogeneous workers.

Figure 6.7 depicts the average time of one iteration, calculated from an execution with 10 repetitions of 10 iterations each. The X-axis indicates the load balancing heuris-

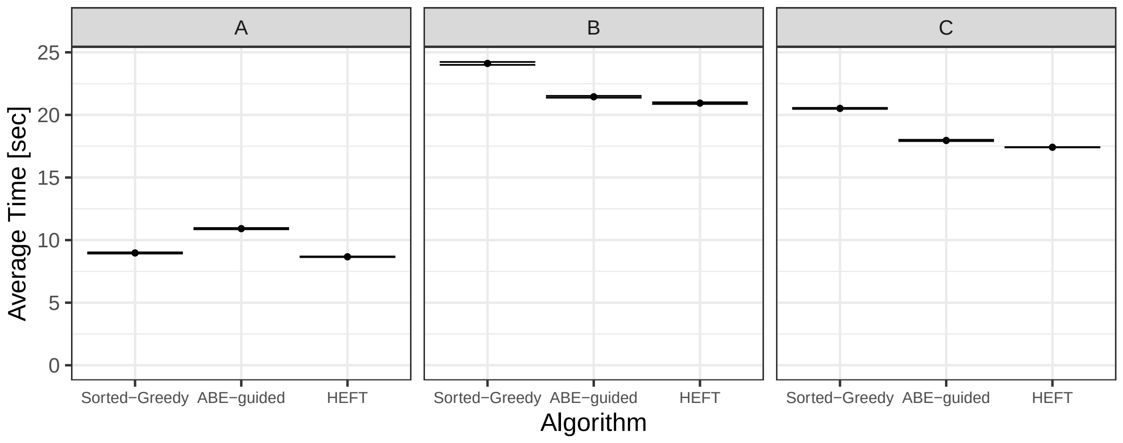
Study case	draco	hype	cei	blaise	Total Cores
A	7	5	2	1	304
B	3	2	1	0	111
C	0	1	2	1	112

Table 6.2 – Setting of the three different case studies for algorithms comparison.

tics, and the Y-axis is the average time, in seconds, that each heuristic presents. Averages are represented by points, while lines represent the error bars for each heuristic. Facets distinguish each different scenario selected to run. We have HEFT as the best average time in all scenarios, presenting an average of ≈8.67 seconds on scenario A, ≈20.9 seconds on scenario B, and ≈17.4 seconds on Scenario C. These averages have as their standard errors ≈0.0268 seconds, ≈0.0684 seconds, and ≈0.0128 seconds, respectively. HEFT solution also presents smaller standard errors when compared with the other heuristics.

Moreover, an interesting observation is in the time difference between B and C scenarios. Even using the same quantity of workers, the third scenario presents a gain of ≈16% compared with the second. This difference confirms the influence of the processor's frequency in the makespan of our application, showing that the machine's combination is another important variable to consider when setting up the environment to run (a characteristic that will be further explored in the next chapter).

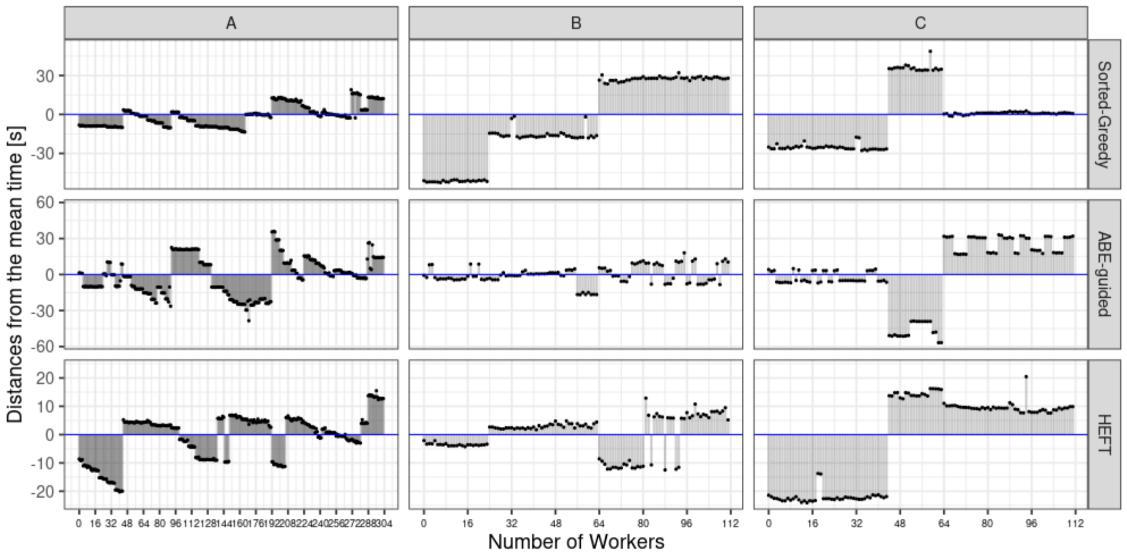
Figure 6.7 – Average time of execution of one iteration in each scheduler running on different heterogeneous configurations.



Source: Author

We also analyze the ideal load balance distance for these solutions. Figure 6.8 depicts the results from the same three previous scenarios of execution (horizontal columns) for the three tested heuristics (vertical columns). The X-axis depicts the number of workers while the Y-axis represents the distance from the ideal meantime, in seconds, with different scales. The blue line on the plot represents the ideal mean time for each scenario. We compute it using a weighted average where we multiply the time spent in each computational resource for its weight and divide the total sum by the sum of weights for each scenario.

Figure 6.8 – The distance from ideal load balance for every worker considering the three different heterogeneous scenarios (columns) and the new scheduling policies (rows).

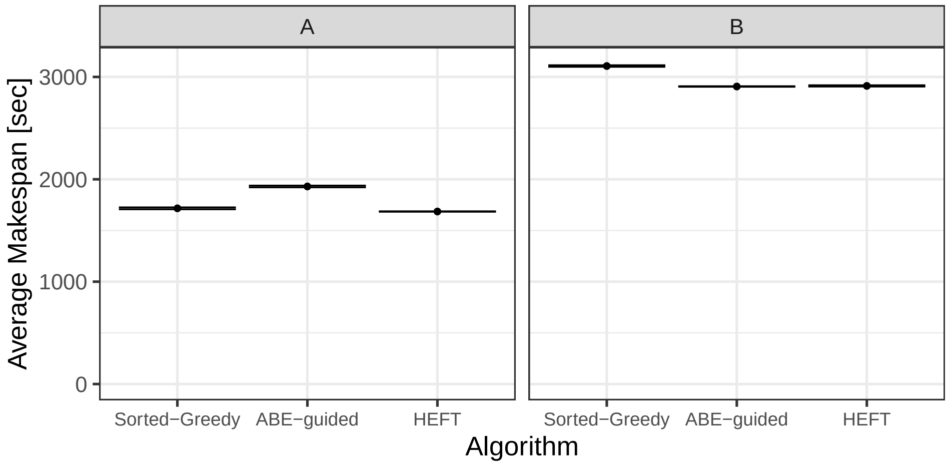


Source: Author

As expected, the Sorted-Greedy heuristic is far from the ideal mean time in these executions. Meanwhile, the results presented by the ABE-guided heuristic were unexpected. Even presenting the best distribution on B scenario, on the C scenario, we can see a distance of almost 60 seconds to the ideal meantime on at least 26 workers. Since ABE should distribute the load focusing on achieving the best makespan, we expected better results from its usage. These results led us to observe that the ABE solution is not exploiting heterogeneous scenarios satisfactorily. We achieve the most stable and best distribution with the HEFT algorithm, where in scenario B, we have more than half of the workers presenting a distance slower than 10 seconds of the ideal mean.

Next, we seek to understand the makespan reduction with this heterogeneous solution. Figure 6.9 depicts the makespan for execution with 100 iterations on A and B scenarios (columns of the plot) previously described, running five repetitions of each heuristic. The X-axis identifies each heuristic, and Y-axis is the average makespan in seconds. In scenario A, using 304 workers, we achieved performances of ≈1930 seconds with ABE-guided and ≈1685 seconds with HEFT. We have more workers than half of CMP cells on the executed model in this scenario. Even with this characteristic, we still present better results than scenario B, with 112 workers. On this second configuration, we have ≈2906 and ≈2912 seconds of average makespan for ABE and HEFT, respectively. When comparing the homogeneous scenario with this same quantity of workers using Sorted-Greedy (average time ≈4649 seconds), we achieved a performance gain of 37% using heterogeneous workers and the HEFT heuristic.

Figure 6.9 – Average makespan with execution of 100 iterations for each scheduler in two different heterogeneous scenarios.



Source: Author

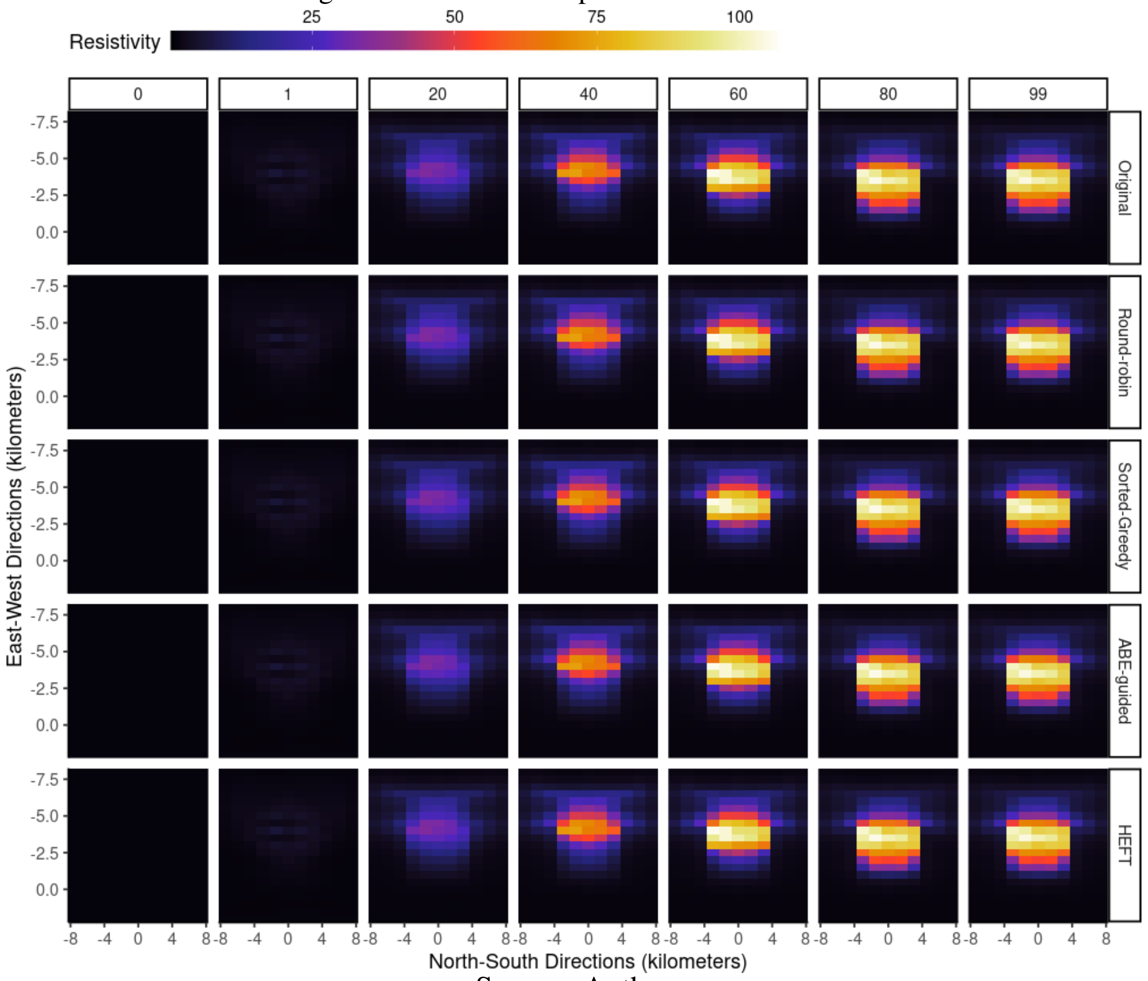
We could achieve a performance gain of 78% by comparing our best heterogeneous case against the Original scheduler. Also, we observed that we achieved a considerable makespan reduction with a different scenario, including those where we do not use all involved resources. This observation, alongside the possibility to have further information to perform a previous scenario of what to expect in real executions, motivated us to develop our next contribution.

6.5 Numerical Accuracy

After all of our changes on the load distribution, we need to guarantee that such modifications did not affect the numerical results. We compare how similar the results from our new scheduling policies are from the Original scheduler. We first carry out a qualitative comparison by visualizing the results. Figure 6.10 depicts the resistivity values (gradient color) for different iterations (columns) of the five scheduling policies (rows).

We center the slice around the body with higher resistivity, using the North-South and East-West direction for a fixed depth that crosses the middle of the body. Visually, numerical results for the scheduling policies are identical. Most of the 3D volume of the resistivity model have similar values because of the homogeneous rock layers with low

Figure 6.10 – Model output from all schedulers.



Source: Author

electrical resistance. Proportionally, the body that geophysicists look for (i.e., petroleum) has a volume much smaller than the whole volume below the prospected seabed area. The known electrical resistance of this body is significantly larger than the surrounding rock layers.

We collect the relative error of the misfit from the execution logs. We repeat the data gathering and the analysis for every iteration and for each scheduling policy. Then, we compute the Peak Signal-To-Noise Ratio (PSNR) (Korhonen; You, 2012) of these differences for each application iteration. Based on the resistivity data, we compute the absolute difference between the numerical results of all our four implemented heuristics against Original. We obtain an infinite PSNR across all iterations for scheduling policies, implying identical numerical results.

7 CONTRIBUTION: CAPACITY PLANNING

As discussed in Chapter 5, we concluded that the application's load remains stable during all its iterations. Each Source-Receiver pair consumes the same processing in the forward region, showing differences only when running on different machines. This conclusion was beneficial for all load balancing solutions, once we consider it as the primary information for choosing the applied load balancing heuristics.

We can also predict application behavior in different settings before real executions with this information regarding time-consuming. To make such a prediction, we need to have a feature that, given a determined quantity of resources and a benchmark, delivers a simulated application behavior in each possible scenario. Such a feature is the primary goal of this contribution. We propose a capacity planning tool that allows users to effectively predict one iteration's behavior with four proposed load balancing strategies without limiting it to one possible scenario.

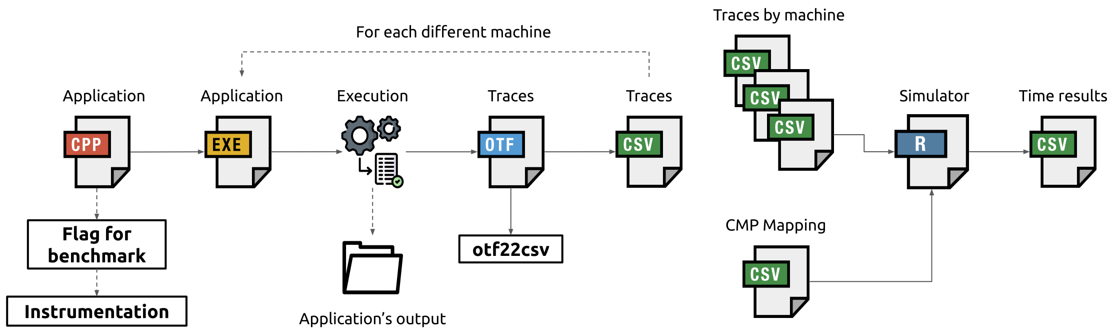
We start this chapter by giving an overview of the methodology implemented to execute this contribution in Section 7.1. We then follow to describe the behavior of our simulation code in Section 7.2. With the code behavior defined, we present the simulation validation on Section 7.3 with real-world executions and statistical evidence. We finish this chapter by presenting how this contribution can be used in different scenarios with a hypothetical test case in Section 7.4.

7.1 Materials and Methods

The methodology applied to this contribution is depicted by Figure 7.1, presenting the steps on the same template previously presented. The first step was focused on understanding the characteristics of the model in the study. Understanding how the application maps the data to the model and how many CMP cells and Source-Receiver pairs we have is essential due to its direct impact on the distribution. We changed our application to enable a flag to gather traces for one execution iteration of each CMP cell. We collected the execution time of each of our three main steps (forward, cholesky and correction) for each cell of the model. This information we will refer as the benchmark of the application. We executed this benchmark in each available resource to be considered in the simulation.

We then developed an R-based solution that applies all studied schedulers for the data collected. The input is composed of a CSV CMP mapping with its respective quantity of S-R pairs and other CSVs containing the forward execution time of each cell in the different resources. We also input which resources are involved in the simulation, informing the quantities of machines and cores. As output, this R code generates the predicted time spent in each algorithm to all possible resource combinations. We can then perform an analytical analysis and generate an interactive plot to gather information from this output.

Figure 7.1 – Figure with methodology



Source: Author

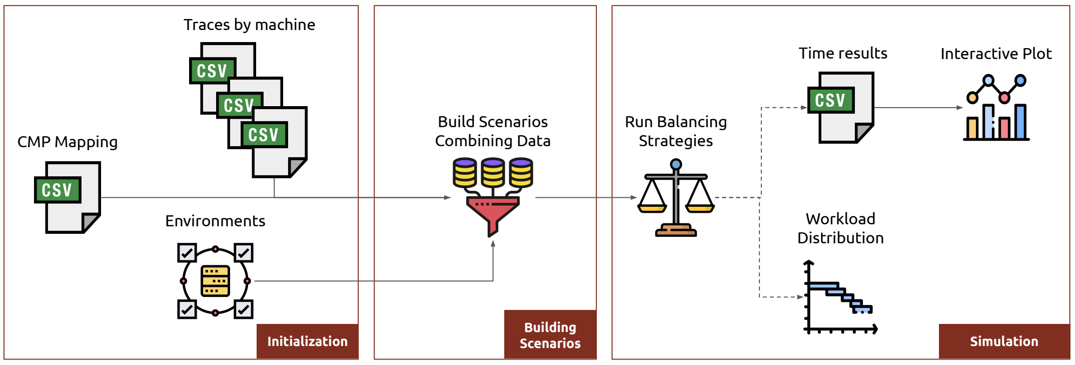
To validate and run the proposed simulator, we used as resource four partitions from the *Parque Computacional de Alto Desempenho* (PCAD) with the characteristics already presented in Table 6.1. We use the same resistivity model and data from the study case presented in Chapter 6.

7.2 Simulating Real Executions

Our simulator comprises three main steps: initialization, building combinations, and simulation. Figure 7.2 helps on explanation by presenting these steps visually. On initialization, we first read the CSV with the CMP mapping of the resistivity model that we aim to simulate. This file is composed of the number identification of the CMP cell and the total amount of S-R pairs linked to it. The application then follows to read the benchmarks of each CMP cell on each different machine. At last, we define the environment being studied, informing how many machines are involved, which partition they are linked to, and how many cores they have.

With this data, the simulator goes through the combination step, where we use a full factorial design to combine machines. It will build different scenarios mixing different quantities of each available resource to run. We go through the simulation step with this all

Figure 7.2 – Workflow diagram of the simulator developed for the capacity planning



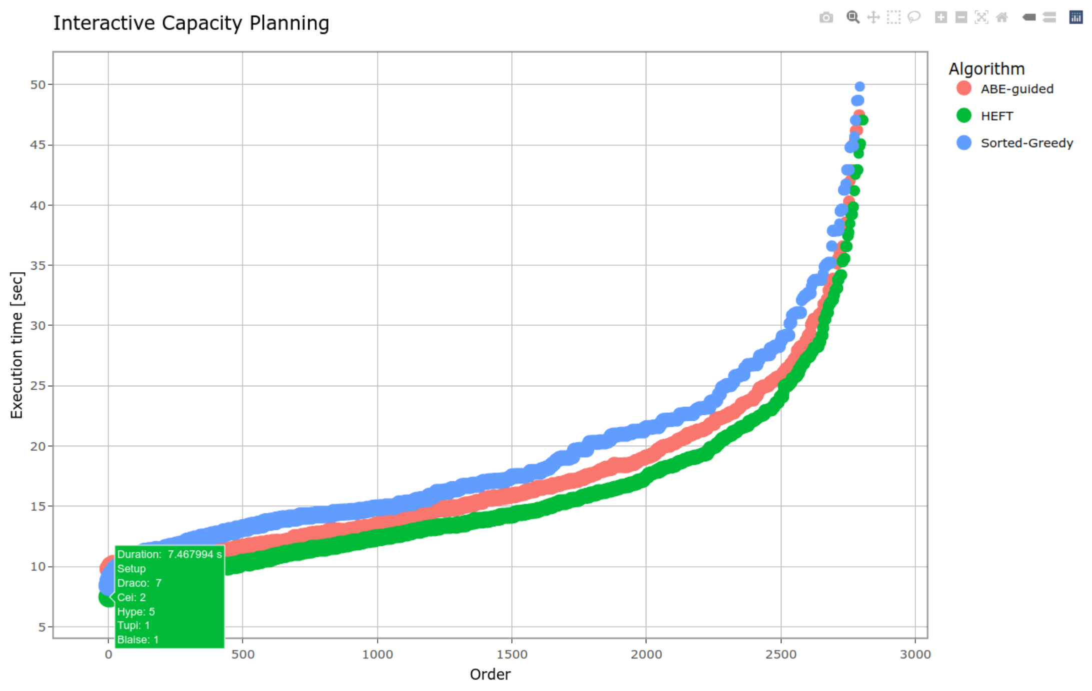
Source: Author

set, where our R application runs the load balancing strategies for each detected scenario. Due to the high number of combinations that we can simulate – which will depend on how many machines we have to combine – we used the `doParallel` R library (THEUSSL; ZEILEIS, 2009) to parallelize the main loop. We can run this simulator using Rscript.

The simulator output two files: the workload distribution (i.e., a CSV mapping file defining which CMP runs in which worker and machine) and the makespan for one iteration with each combination. From these collected makespans, we can build an interactive plot using the Plotly tool (INC., 2015). An example of this output can be found at https://jessicadagostini.gitlab.io/master-degree/interactive-capacity-planning.html.

Figure 7.3 depicts a screenshot from this interactive application. We can extract many information from this tool. First, the executions on the plot are sorted according to its makespan, going from the faster to the slowest (X-axis). Each point in the graph represents a different simulation scenario with its makespan in seconds (Y-axis). Hovering each point will inform us which partitions are involved in that result, how many nodes of each resource it was used, and the makespan of this one iteration simulated. The different colors in the graph identify the four load balancing heuristics executed, and the size of each point indicates how many cores were involved in each scenario. In the example of the Figure 7.3, we have a scenario where we used seven machines from Draco partition, five from Hype, one from Tupi, two from Cei, and one from Blaise. This machine's combination resulted in a makespan of ≈7.46 seconds for one iteration, which is one of the best results we can achieve.

Figure 7.3 – Screen capture of the interactive plot with all possible scenarios of machine's combination. The colors of the plot represent the different algorithms, and the different size of points indicates how many cores each scenario is composed of.



Duration: 7.467994 s
Setup
Draco: 7
Cei: 2
Hype: 5
Tupi: 1
Blaise: 1

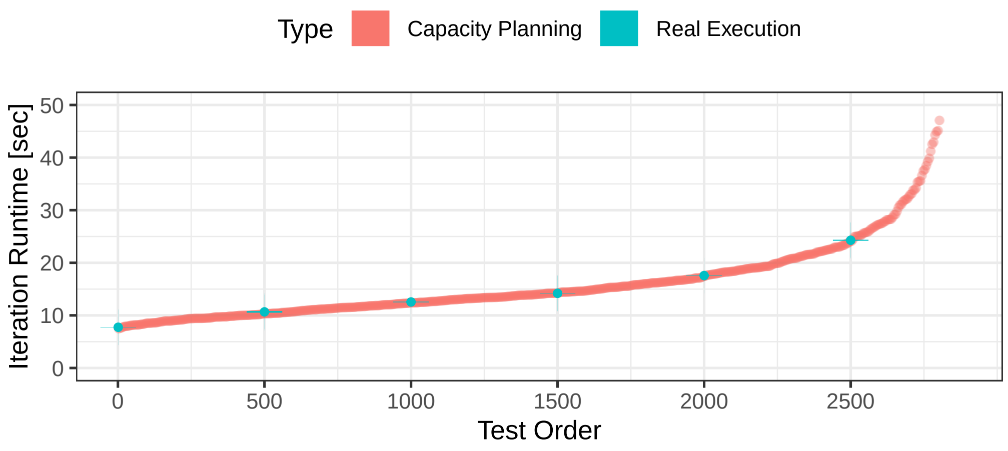
Source: Author

7.3 Simulation Validation

To present accurate simulation results, we aim to validate our simulation code. First, we run our simulator giving as input the environment characteristics as we found on PCAD. We define our environment containing seven Dracos, five Hypes, two Ceis, one Tupi, and one Blaise machine. From the simulation output, we organize from the fastest to the slowest scenarios and choose seven scenarios to perform real executions. We run ten repetitions of each chosen setting, with five iterations each. Figure 7.4 depicts the output for the HEFT algorithm. We have the test order on the X-axis, from 1 to the total number of executions ordered as previously described. On the Y-axis, we have the iteration simulated runtime in seconds. The pink color represents the simulated makespan, and the blue dots are the average runtime of 50 iterations for each of the chosen validation scenarios.

The plot of Figure 7.4 depicts that the simulator's predicted time is accurate with the real executions. Analyzing closer two scenarios, represented by Test Order 0 and Test Order 1500, we have a simulated time equal to 7.47s and an average makespan of 7.74s for the former, and the same 14.2 seconds for simulated time and average makespan for

Figure 7.4 – Validation of results from our R simulator code. We validate our simulation with HEFT algorithm. The blue points represent the average runtime of real executions.



Source: Author

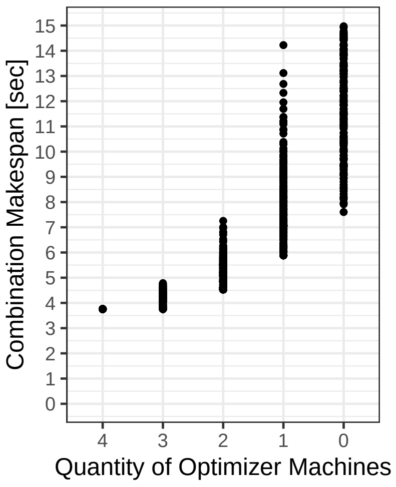
the second.

7.4 Hypothetical Test

Alongside helping to choose the best combination of existing resources to run the application, our tool can model a hypothetic case with non-existing machines. Suppose that the system administrators of our cluster received an offer to buy new powerful machines, which we will call here Optimizer. These machines are two times faster than our best machines nowadays and are composed of 64 cores each. The administrator asked us how many machines we need to achieve a better performance in our application. Using our simulator code, we can give a more accurate answer to this question. We used real traces from our existing machines and generated a synthetic trace for the new one based on our best existing machine, Blaise. We know that administrators have a budget to buy up to four new machines. We ran our simulator and collected the results. Figure 7.5 depicts the iterations makespan in seconds (Y-axis) achieved with the simulation of HEFT to each possible scenario (points) varying the quantity of Optimizer nodes (X-axis).

With four Optimizer machines, all 1375 scenarios presented the same makespan (3.76 seconds). This can be explained due to the high quantity of cores that these combinations have, which is equal or bigger the half of total quantity of CMP cells in the test case. Considering only the cores from Optimizer machines, we have 256 cores, which led us to be limited in time to the bigger CMP of the test case (as previously discussed on Chapter 6). Using three Optimizer machines we still have scenarios where we have high

Figure 7.5 – Makespan with combination scenarios involving different quantities of Optimizer machines running HEFT distribution heuristic.



Source: Author

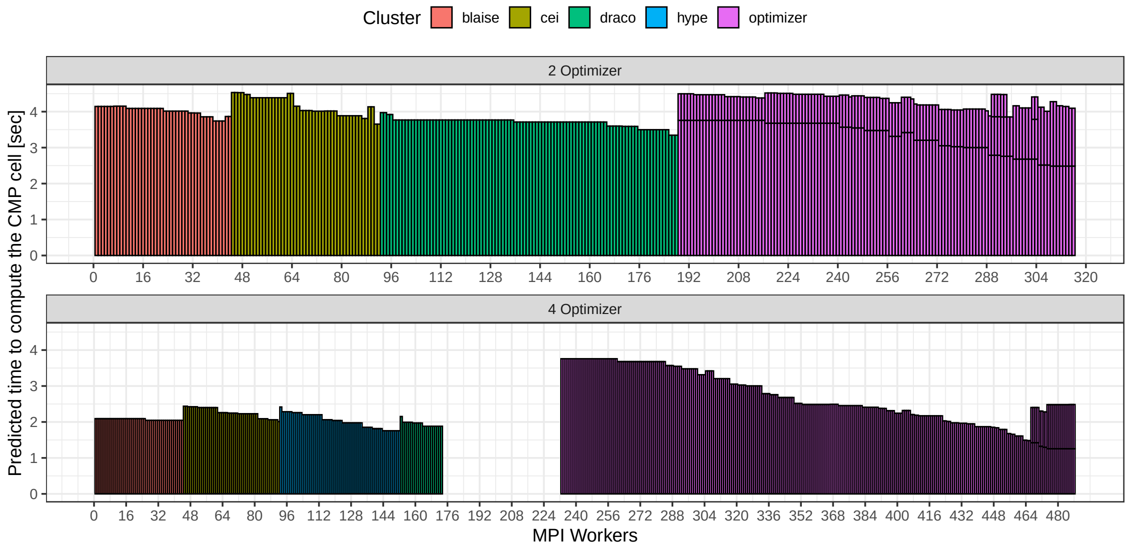
quantities of cores. The scenarios with two Optimizer present significant results, with an median makespan of 5.20 seconds. We can observe an expressive speed difference when comparing no usage of Optimizer machines from those using at least two nodes.

Figure 7.6 helps visualize the resource usage with a scenario using two Optimizer machines and the other with four. The time spent on each worker is depicted on Y-axis, being X-axis responsible for identifying each different worker. Each column represents different CMP cells allocated to each worker, while the colors of the plot indicate which partition each worker represents. We present the distribution result of the HEFT algorithm. In the four-machines scenario, we identify 59 idle workers, even some workers received more than one cell to compute. This behavior happens because, when estimating the time spent on these idle workers, the HEFT algorithm concluded that assigning the tasks to other workers performed better results than those who do not have any load to compute. Once workers who did not receive the load to compute are from machines already present on the cluster, this solution did not make the best use of resources. In the two-machines scenario, we do not see any idle workers. With all these analyses, we can conclude that we can improve the application's speed by having at least two Optimizer machines added to our existing cluster.

7.5 Final Remarks

From all above discussed, we presented a tool that allows different conclusions from the presented data. First, we can detect which of the three heterogeneous distribu-

Figure 7.6 – Examples of CMP allocation in two scenarios, one using two Optimizer machines and other using four new machines. We can identify that, when using four machines, some workers did not receive any load to compute.



Source: Author

tions better fit each specific scenario. Second, we can identify which machines combination best fits our ready-to-use resources. Commonly, HPC clusters are used for many users simultaneously, making some machines already allocated to other jobs when we need them. With this tool, we can choose the scenario that delivers the best makespan with the resources available at that moment. Once the cluster characteristics are set up before running the code, we can apply this solution to any cluster scenario. The only requirement is to execute a previous benchmark from the application in the target resources.

This feature allows application users to plan executions with precise information regarding how much time they will spend to have results. It is important to notice that the application is iterative and, for each study case, it takes different quantities of iterations to converge. However, previously knowing the time spent by one iteration allows users to better view its execution and better use their resources. Moreover, this solution can also help system administrators with cluster acquisitions plans. If we know an expected behavior of a new desired machine, we can use this simulator to help decide how many new machines we need to have a reasonable makespan from this problem.

8 CONCLUSION

The industrial investment in electromagnetic surveying to recognize seafloor aspects and help find oil and gas reservoirs has driven innovations in the data acquisition of these methods. Consequently, it generates a large amount of data that presents a significant computational challenge to recover the resistivity of the regions in analysis throughout inversion processes, once that 2D or 3D inversion methods require high computational usage. Thus, a modern inversion code should apply parallel strategies to run such applications in a reasonable time, taking care of all aspects that compose these parallel execution.

This work investigates the computing performance of an MPI-based parallel application that carries out the data inversion of the marine Controlled Source Electromagnetic method. Such an application apply a Common Mid-Point (CMP) technique, which consists of a mapping combination based on the common geographical mid-point of Source-Receivers pairs in a collection of CMP cells given by a resistivity model. We perform analysis with a synthectic workload to understand the application's behavior, the composing of its load, and the load distribution strategy used.

We identify an issue regarding load balancing on the application. The original strategy used to distribute the load across the parallel workers considered the total number of CMP cells to divide throughout the workers. However, our analysis presented that the actual load of the application is the amount of S-R pairs linked to each CMP cell. The original distribution strategy divided the wrong load grain, generating the load imbalance.

From these conclusions, we implement four load balancing strategies to mitigate the problem, two of them focusing on homogeneous platforms and the other two focused on heterogeneous platforms. On homogeneous scenarios, we apply Round Robin and Sorted-Greedy heuristics. We achieved a performance gain of 41% with the Sorted-Greedy heuristic. Regarding heterogeneous solutions, we implemented an Area Bound Estimation Guided and Heterogeneous Earliest Finish Time (HEFT) approaches, where we adapted both solutions to our needs. For the ABE-guided solution, we implemented the linear programming in such a format that we could also extract the vector of distribution used to achieve the best theoretical makespan. We also changed the HEFT algorithm to only consider the time spent in each heterogeneous environment to allocate new tasks to a worker. We tested these two strategies in three heterogeneous scenarios, where we achieved a performance gain of 78% compared to the original scheduler.

Moreover, due to the static characteristic of our load, we propose a tool that helps in capacity planning for this application. We implemented our four load balancing heuristics in an R-based code which simulates, from a given input containing the benchmark of any study case and the number of machines involved, a diverse set of possible machine combinations, and its makespan for one iteration of the method. Our tool delivers an accurate simulation, which enables us to perform such simulation before real executions of the inversion application. By this simulation, we can further predict the best execution scenario with the available resources we have at that moment.

As future work, we aim to port part of the application's code to GPU accelerators. Once this method applies many matrix operations, we can gain even more performance on executing these operations on GPUs. To make such improvement is necessary a better understanding of the numerical methods applied by the inversion process to apply modifications in a correct way.

8.1 Publications

- **DAGOSTINI, J.I.**;DA SILVA, H. C. P.; PINTO, V. G.;VELHO, R. M.; GASTAL, E. S. L.;SCHNORR, L. M.. "Improving Workload Balance of a Marine CSEM Inversion Application," 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2021, pp. 704-713. DOI: <https://doi.org/10.1109/IPDPSW52791.2021.00107>.
- **DAGOSTINI, Jessica**; PINTO, Vinicius; SCHNORR, Lucas. Aplicação de Método Guloso para Balanceamento de Carga em uma Aplicação de Exploração Eletromagnética. In: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS), 21., 2021, Evento Online. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2021 . p. 87-88. ISSN 2595-4164. DOI: <https://doi.org/10.5753/eradrs.2021.14782>.

REFERENCES

ALAKEEL, A.; ALAKEEL, A. M. A guide to dynamic load balancing in distributed computer systems. **IJCSNS International Journal of Computer Science and Network Security**, v. 10, 2010. Available from Internet: <https://www.researchgate.net/publication/268200851>.

ALAPPAT, C. et al. A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. **ACM Trans. Parallel Comput.**, Association for Computing Machinery, New York, NY, USA, v. 7, n. 3, jun 2020. ISSN 2329-4949. Available from Internet: <https://doi.org/10.1145/3399732>.

AMESTOY, P. R. et al. Mumps: a general purpose distributed memory sparse solver. In: SPRINGER. **International Workshop on Applied Parallel Computing**. [S.l.], 2000. p. 121–130.

BALAY, S. et al. **PETSc Web page**. 2019. <https://www.mcs.anl.gov/petsc>.

CASANOVA, H.; LEGRAND, A.; ROBERT, Y. **Parallel Algorithms**. [S.l.: s.n.], 2008. ISBN 9780429148484.

CASTILLO-REYES, O.; PUENTE, J. de la; CELA, J. M. Petgem: A parallel code for 3d csem forward modeling using edge finite elements. **Computers and Geosciences**, Elsevier Ltd, v. 119, p. 123–136, 10 2018. ISSN 00983004.

CONSTABLE, S. Ten years of marine csem for hydrocarbon exploration. **GEOPHYSICS**, v. 75, n. 5, p. 75A67–75A81, 2010.

CONSTABLE, S.; SRNKA, L. J. An introduction to marine controlled-source electromagnetic methods for hydrocarbon exploration. **GEOPHYSICS**, v. 72, n. 2, p. WA3–WA12, 2007. Available from Internet: <https://doi.org/10.1190/1.2432483>.

CORRÊA, J.; RÉGIS, C. T. 1d CMP inversion of MCSEM data to create a 3d geoelectrical model. In: **SEG Technical Program Expanded Abstracts 2017**. [S.l.]: Society of Exploration Geophysicists, 2017.

da Piedade, A. A. et al. Computational cost comparison between nodal and vector finite elements in the modeling of controlled source electromagnetic data using a direct solver. **Computers and Geosciences**, v. 156, p. 104901, 2021. ISSN 0098-3004.

DAGOSTINI, J. I. et al. Improving workload balance of a marine csem inversion application. In: **2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.: s.n.], 2021. p. 704–713.

DAVIS, T. A.; HU, Y. The university of florida sparse matrix collection. **ACM Trans. Math. Softw.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 1, dec 2011. ISSN 0098-3500. Available from Internet: <https://doi.org/10.1145/2049662.2049663>.

DONGARRA, J. et al. With extreme computing, the rules have changed. **Computing in Science Engineering**, v. 19, n. 3, p. 52–62, 2017.

FOSTER, I.; FOSTER, J. **Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering**. Addison-Wesley, 1995. (Literature and Philosophy). ISBN 9780201575941. Available from Internet: <https://books.google.com.br/books?id=r5JsQgAACAAJ>.

GAREY, M. R.; JOHNSON, D. S. **Computers and intractability**. [S.l.]: freeman San Francisco, 1979.

GEORGE, E. et al. **Statistics for experimenters: design, innovation, and discovery**. [S.l.]: Wiley New York, 2005.

GOLUB, G.; ORTEGA, J. M. Chapter 1 - the world of scientific computing. In: GOLUB, G.; ORTEGA, J. M. (Ed.). **Scientific Computing**. San Diego: Academic Press, 1993. p. 1–14. ISBN 978-0-12-289253-0. Available from Internet: <https://www.sciencedirect.com/science/article/pii/B9780122892530500051>.

GRAYVER, A. V.; BüRG, M. Robust and scalable 3-D geo-electromagnetic modelling approach using the finite element method. **Geophysical Journal International**, v. 198, n. 1, p. 110–125, 04 2014. ISSN 0956-540X. Available from Internet: <https://doi.org/10.1093/gji/ggu119>.

GRAYVER, A. V.; STREICH, R.; RITTER, O. Three-dimensional parallel distributed inversion of CSEM data using a direct forward solver. **Geophysical Journal International**, Oxford Academic, v. 193, n. 3, p. 1432–1446, 2013. ISSN 0956540X.

INC., P. T. **Plotly: Collaborative data science**. Montreal, QC: Plotly Technologies Inc., 2015. Available from Internet: <https://plot.ly>.

KEY, K. 1D inversion of multicomponent, multifrequency marine CSEM data: Methodology and synthetic studies for resolving thin resistive layers. **Geophysics**, v. 74, n. 2, p. F9–F20, 02 2009. ISSN 0016-8033.

KEY, K. Marine em inversion using unstructured grids: A 2d parallel adaptive finite element algorithm. In: . [S.l.]: Society of Exploration Geophysicists, 2012. p. 574–578. ISBN 9781622769452.

KEY, K. Mare2dem: A 2-d inversion code for controlled-source electromagnetic and magnetotelluric data. **Geophysical Journal International**, Oxford University Press, v. 207, p. 571–588, 10 2016. ISSN 1365246X.

KEY, S. W.; KRIEG, R. D. Comparison of finite-element and finite-difference methods**this work was supported by the united states atomic energy commission. In: FENVES, S. J. et al. (Ed.). **Numerical and Computer Methods in Structural Mechanics**. Academic Press, 1973. p. 337–352. ISBN 978-0-12-253250-4. Available from Internet: <https://www.sciencedirect.com/science/article/pii/B9780122532504500191>.

KNüPFER, A. et al. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. **Tools for High Performance Computing 2011**, Springer Berlin Heidelberg, p. 79–91, 2012. Available from Internet: <http://dx.doi.org/10.1007/978-3-642-31476-6_7>.

Korhonen, J.; You, J. Peak signal-to-noise ratio revisited: Is simple beautiful? In: **Int. WS on Quality of Multimedia Exp.** [S.l.: s.n.], 2012. p. 37–38.

LEVENBERG, K. A method for the solution of certain non-linear problems in least squares. **Quarterly of Appl. Math.**, v. 2, n. 2, p. 164, 1944.

LONG, Z. et al. Parallelized 3-D CSEM Inversion With Secondary Field Formulation and Hexahedral Mesh. **IEEE Trans. on Geo. and Rem. Sens.**, Institute of Electrical and Electronics Engineers (IEEE), v. 58, n. 10, p. 6812, 2020. ISSN 0196-2892.

MARQUARDT, D. W. An algorithm for least-squares estimation of nonlinear parameters. **Journal of the society for Industrial and Applied Mathematics**, SIAM, v. 11, n. 2, p. 431–441, 1963.

MERTENS, S. Number partitioning. **Computational Complexity and Statistical Physics**, Oxford University Press on Demand, p. 125, 2006.

MITTET, R. High-order finite-difference simulations of marine csem surveys using a correspondence principle for wave and diffusion fields. **Geophysics**, Society of Exploration Geophysicists, v. 75, n. 1, p. F33–F50, 2010.

NESI, L. L.; LEGRAND, A.; SCHNORR, L. M. Exploiting system level heterogeneity to improve the performance of a geostatistics multi-phase task-based application. In: **50th International Conference on Parallel Processing**. New York, NY, USA: Association for Computing Machinery, 2021. (ICPP 2021). ISBN 9781450390682. Available from Internet: <https://doi.org/10.1145/3472456.3472516>.

NOCEDAL, J.; WRIGHT, S. J. **Numerical Optimization**. [S.l.: s.n.], 2006.

PEARCE, O. et al. Quantifying the effectiveness of load balance algorithms. In: **Proceedings of the 26th ACM international conference on Supercomputing**. [S.l.: s.n.], 2012. p. 185–194.

PETHICK, A.; HARRIS, B. Macro-parallelisation for controlled source electromagnetic applications. **Journal of Applied Geophysics**, v. 124, p. 91–105, 2016. ISSN 0926-9851. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0926985115300641>.

PINTO, V. G. **Performance Analysis Strategies for Task-based Applications on Hybrid Platforms**. Thesis (PhD) — Université Grenoble Alpes; Universidade Federal do Rio Grande do Sul (Brésil), 2018.

R Core Team. **R: A Language and Environment for Statistical Computing**. Vienna, Austria, 2020. Available from Internet: <https://www.R-project.org/>.

RAUBER, T.; RÜNGER, G. Parallel programming models. In: ____. **Parallel Programming: for Multicore and Cluster Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 105–167. ISBN 978-3-642-37801-0. Available from Internet: <https://doi.org/10.1007/978-3-642-37801-0_3>.

REN, Z. et al. A goal-oriented adaptive finite-element approach for plane wave 3-D electromagnetic modelling. **Geophysical Journal International**, v. 194, n. 2, p. 700–718, 05 2013. ISSN 0956-540X. Available from Internet: <https://doi.org/10.1093/gji/ggt154>.

SCHNORR, L. M. **OTF2Utils**. Rio Grande do Sul, BR: [s.n.], 2019. Available from Internet: <https://github.com/schnorr/otf2utils>.

Silva Crepaldi, J. L.; Pereira Buonora, M. P.; FIGUEIREDO, I. Fast marine CSEM inversion in the CMP domain using analytical derivatives. **Geophysics**, Society of Exploration Geophysicists, v. 76, n. 5, p. F303–F313, sep. 2011. ISSN 168033. Available from Internet: <https://doi.org/10.1190/geo2010-0237.1>.

SILVA, N. V. da et al. A finite element multifrontal method for 3d csem modeling in the frequency domain. **GEOPHYSICS**, v. 77, n. 2, p. E101–E115, 2012.

SOMMER, M. et al. GPU parallelization of a three dimensional marine CSEM code. **Computers & Geosciences**, Elsevier BV, v. 58, p. 91–99, aug. 2013.

SOUZA, V. C. T. de. **Modelagem Numérica de Dados mCSEM 3D Usando Computação Paralela**. 110 p. Thesis (PhD) — Universidade Federal do Pará, 2007. Available from Internet: <http://repositorio.ufpa.br:8080/jspui/handle/2011/5673>.

STREICH, R. 3D finite-difference frequency-domain modeling of controlled-source electromagnetic data: Direct solution and optimization for high accuracy. **GEOPHYSICS**, Society of Exploration Geophysicists, v. 74, n. 5, p. F95–F105, 2009. ISSN 0016-8033.

TESSER, R. K. **A Simulation Workflow to Evaluate the Performance of Dynamic Load Balancing with Over-decomposition for Iterative Parallel Applications**. Thesis (PhD) — Universidade Federal Do Rio Grande Do Sul, 2018.

THEUSSL, S.; ZEILEIS, A. Collaborative software development using r-forge. special invited paper on" the future of r". **The R Journal**, The R Foundation for Statistical Computing, v. 1, n. 1, p. 9–14, 2009.

TOPCUOGLU, H.; HARIRI, S.; WU, M.-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. **IEEE Transactions on Parallel and Distributed Systems**, v. 13, n. 3, p. 260–274, 2002.

TROBEC, R. et al. **Introduction to Parallel Computing: From Algorithms to Programming on State-of-Art Platforms**. [S.l.]: Springer, 2018.

WANG, F.; MORTEN, J. P.; SPITZER, K. Anisotropic three-dimensional inversion of CSEM data using finite-element techniques on unstructured grids. **Geophysical Journal International**, Oxford University Press (OUP), v. 213, n. 2, p. 1056, 2018.

WICKHAM, H. **ggplot2: Elegant Graphics for Data Analysis**. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. Available from Internet: <https://ggplot2.tidyverse.org>.

WICKHAM, H. et al. Welcome to the tidyverse. **Journal of Open Source Software**, v. 4, n. 43, p. 1686, 2019.

WILT, M.; ALUMBAUGH, D. Electromagnetic methods for development and production: State of the art. **The Leading Edge**, v. 17, n. 4, p. 487–487, 1998. Available from Internet: <https://doi.org/10.1190/1.1437997>.

YANG, P. Boost the efficiency of 3d csem modelling using graphics processing units. In: EUROPEAN ASSOCIATION OF GEOSCIENTISTS & ENGINEERS. **82nd EAGE Annual Conference & Exhibition**. [S.l.], 2021. v. 2021, n. 1, p. 1–5.

ZHAO, N.; YU, G.; QIN, C. Three-dimensional mcsem inversion based on parallel goal-oriented adaptive finite element method for large scale problem. In: **24th EM Induction Workshop**. [S.l.: s.n.], 2018. v. 2018, n. 1, p. 1–4.

APÊNDICE A — RESUMO EXPANDIDO EM PORTUGUÊS

In this appendix, we present a summary of this Master Dissertation, in Portuguese, as required by the PPGC Graduate Program in Computing at UFRGS.

Neste apêndice, apresentamos um resumo desta Dissertação de Mestrado, em Português, conforme determinado pelo Programa de Pós-Graduação em Computação da UFRGS.

A.1 Motivação

Um dos principais desafios da indústria de petróleo e gás é mapear corretamente as áreas no oceano onde há maior probabilidade de ocorrência de petróleo e gás. A exploração física de petróleo geralmente requer quantias de dinheiro na ordem de milhões de dólares. Para ajudar nessa tarefa, cientistas e empresas recorrem a simulações computacionais para reduzir a incidência de operações de perfuração malsucedidas, economizando tempo e dinheiro. Vários métodos numéricos e computacionais podem ser aplicados a esta exploração envolvendo dados eletromagnéticos e sísmicos.

Um dos levantamentos eletromagnéticos para simular e investigar computacionalmente as terras oceânicas é o marine Controlled Source Electromagnetic (mCSEM) (SOUZA, 2007), Método Eletromagnético Marinho de Fonte Controlada, em Português. Este método utiliza receptores marítimos fixados em uma região específica do fundo oceânico para coletar dados eletromagnéticos dessas regiões. Um navio carregando um emissor eletromagnético se move nesta área predeterminada e os receptores medem o campo eletromagnético da região depois que as ondas eletromagnéticas atravessam a subsuperfície sob o fundo do mar e refletem. Este levantamento é aplicado para reconhecer os aspectos do fundo do mar, fornecendo assim informações relevantes sobre a resistividade da região explorada.

Para interpretar esses dados, precisamos usar métodos numéricos que aplicam uma inversão matemática para aproximar rapidamente um modelo correspondente da subsuperfície em análise e, assim, ter uma melhor compreensão dos elementos que compõem a região. Tal modelo de subsuperfície é referenciado aqui como o modelo de resistividade e tem uma estimativa inicial das resistividades da região em análise. A técnica de inversão de dados é comumente usada em aplicações computacionais de petróleo/gás, pois estes materiais possuem assinaturas eletromagnéticas bem conhecidas em comparação com outros materiais (KEY, 2009).

Somado aos esforços relacionados aos métodos numéricos, uma perspectiva computacional em qualquer aplicação científica é sempre essencial. Não há área complexa da ciência que não use computadores para modelagem (GOLUB; ORTEGA, 1993). Dessa forma, é essencial que não apenas os matemáticos modelem corretamente as equações numéricas - necessárias para resolver cada problema específico -, mas também que os cientistas da computação modelem a aplicação de acordo com seus conhecimentos sobre desempenho e computabilidade. Buscamos trazer uma perspectiva computacional propondo e executando uma avaliação prática de desempenho utilizando ferramentas de última geração para a aplicação de inversão mCSEM estudada.

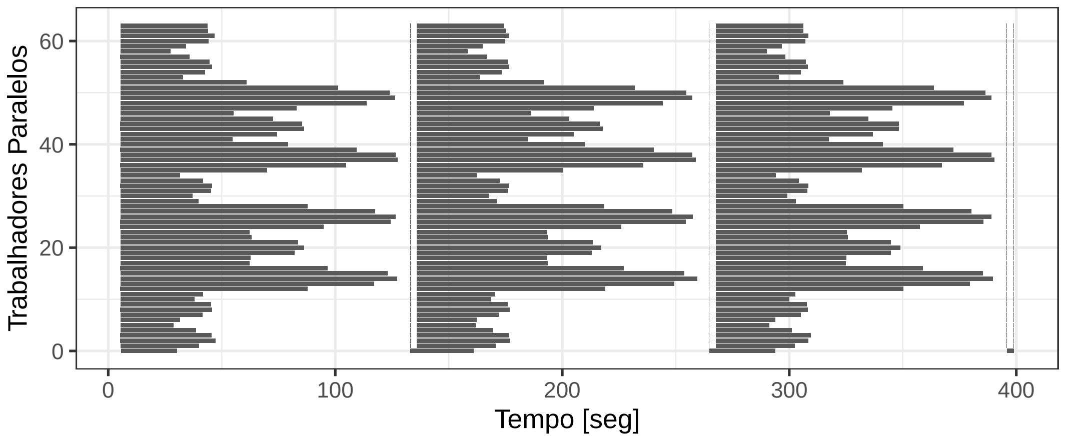
Neste trabalho, nós estudamos uma aplicação de inversão de dados que usa equações diretas finitas e a técnica de ponto médio comum, do inglês *common mid-point* para lidar com dados coletados do método mCSEM. Essa aplicação mapeia os dados do levantamento eletromagnético para os pontos do modelo de resistividade inicial na inicialização. Em seguida, em uma fase que chamamos de *forward*, a aplicação executa equações numéricas baseadas nos métodos de Maxwell e Levenberg-Marquardt para evoluir o modelo de forma iterativa. Essas equações são a etapa mais computacionalmente cara do processo de inversão, uma vez que é nela em que se executa a maioria das equações numéricas do método. Para executar essa fase um tempo razoável, precisamos dividir a carga de trabalho da aplicação entre trabalhadores paralelos.

Clusters de alto desempenho são cruciais para as velocidades de processamento exigidas desses volumes significativos de dados e processos. Nesses ambientes, usamos máquinas que possuem um processo poderoso para executar aplicações caras. No entanto, apenas utilizar esses ambientes não é suficiente para mitigar os custos computacionais. Precisamos usar as máquinas de cluster da melhor forma possível, garantindo que todos os recursos estejam e equilibrados. Cada recurso deve receber carga suficiente para computar em uma duração semelhante. Essa distribuição dá a aplicação uma característica de balanceamento de carga, um ponto chave para sua velocidade.

As técnicas de balanceamento de carga garantem o uso eficiente de recursos em aplicações de alto desempenho, que aumentam a velocidade e a escalabilidade destes e combatem o problema de desbalanceamento. Garantir essa eficiência é fundamental para as aplicações paralelas de hoje, pois pode significar que milhares de núcleos são alocados para execuções, mas podem ficar ociosos (PEARCE et al., 2012). Há um número considerável de estratégias de balanceamento de carga estudadas na literatura. Aplicar tais soluções de balanceamento de carga a problemas do mundo real pode contribuir para a evolução desta área de pesquisa.

Procurar um desequilíbrio de carga em um aplicação requer instrumentação do código e adição de marcas de tempo para saber quanto tempo cada trabalhador paralelo leva para executar uma tarefa. Quando os rastros de execução mostram que alguns trabalhadores demoram mais para concluir uma tarefa do que outros, temos um desequilíbrio de carga. A Figura A.1 demonstra esse conceito com nossa aplicação. Descrevemos os resultados das três primeiras iterações de uma execução com 100 iterações, apresentando o tempo de execução em segundos no eixo X e os trabalhadores paralelos no eixo Y. Nesses rastros, o primeiro trabalhador termina suas tarefas $2,3\times$ mais rápido que o décimo nono trabalhador. Esses resultados nos motivam a aplicar melhorias computacionais em nossa aplicação investigada.

Figura A.1 – Exemplo de execução com distribuição desbalanceada, gerado a partir de dados coletados da execução da aplicação em estudo.



Fonte: Autor

Melhorias nas aplicações mCSEM têm alguns outros relatos na literatura. (ZHAO; YU; QIN, 2018) e (SILVA et al., 2012) apresentaram trabalhos sobre melhorias numéricas no processo de inversão. O primeiro aplicou um algoritmo de campo total ao processo de inversão, usando um modelo em camadas 3D e a biblioteca METIS para o particionamento do domínio. A segunda dividiu a aplicação de inversão em dois componentes e utilizou um método de particionamento multinível, aplicando a biblioteca MUMPS para paralelização e um código MPI+OpenMP híbrido. Caso contrário, (LONG et al., 2020) e (da Piedade et al., 2021) discutem os benefícios de usar a biblioteca Pardiso para obter ganhos de velocidade computacional. Paradiso é um software para resolver grandes sistemas lineares esparsos em memória compartilhada e distribuída (ALAPPAT et al., 2020). No entanto, nenhum trabalho focou em estratégias de balanceamento de carga para melhorar o desempenho. Aplicamos heurísticas de balanceamento de carga de última geração em nosso trabalho e propusemos um planejamento de capacidade com tais estratégias.

A.2 Contribuições

Encontramos espaço para melhorias em nossa aplicação com uma análise de desequilíbrio de carga. Identificamos que o problema provinha na política de distribuição de carga de trabalho original, impactando diretamente na fase de *forward* paralela. A partir disso, implementamos e avaliamos heurísticas de balanceamento de carga para melhorar a aplicação, considerando a necessidade de distribuir tais dados de acordo com o mapa de resistividade fornecido. Nosso objetivo foi de aumentar a velocidade da aplicação usando melhor os recursos computacionais disponíveis e melhorando sua escalabilidade em clusters homogêneos e heterogêneos. Além disso, devido à característica estática da carga em cada ambiente onde foi executada, também propomos um planejamento de capacidade, do inglês *capacity planning*, para auxiliar no planejamento de execução em clusters maiores e heterogêneos. Desenvolvemos uma ferramenta interativa que ajuda a determinar quanto tempo o programa levará para ser executado em um conjunto diversificado de combinações de máquinas. Tal recurso traz um passo versátil e decisivo na execução dessa aplicação de inversão de dados, permitindo-nos antecipar como nossa execução se comportará em uma determinada configuração. Tais previsões suportam um melhor escalonamento de recursos.

Em resumo, estas são as principais contribuições:

- Propomos uma análise de desempenho, com foco na compreensão do código da aplicação e sua instrumentação, além da identificação das suas principais regiões de execução de acordo com o método numérico aplicado. Essa análise nos levou a identificar um desequilíbrio de carga na aplicação.
- Estudamos e implementamos heurísticas de balanceamento de carga para ambientes homogêneos e heterogêneos, ampliando as possibilidades de execução em diferentes cenários. Em cenários homogêneos, aplicamos as heurísticas Round Robin e Sorted-Greedy, onde alcançamos um ganho de desempenho de 41% com a heurística Sorted-Greedy. No que diz respeito às soluções heterogêneas, implementamos as abordagens Area Bound Estimation (ABE) Guided e Heterogeneous Earliest Finish Time (HEFT), e alcançamos 78% de melhoria de desempenho com o algoritmo

HEFT.

- Desenvolvemos um planejamento de capacidade para a aplicação, baseado em dados de tempo de execução da mesma em diferentes máquinas e na plataforma de destino onde ela pode ser executada.

A.3 Conclusões

Neste trabalho investigamos o desempenho computacional de uma aplicação paralela baseada em MPI que realiza a inversão de dados do método marine Controlled Source Electromagnetic. Realizamos análises com uma carga de trabalho sintética para entender o comportamento da aplicação, a composição de sua carga e a estratégia de distribuição de carga utilizada. A partir disso, identificamos um problema relacionado ao balanceamento de carga, onde a estratégia originalmente usada considerou o grão de carga errado, gerando o desequilíbrio de carga.

A partir dessas conclusões, implementamos quatro estratégias de balanceamento de carga para mitigar o problema, duas delas com foco em plataformas homogêneas e as outras duas com foco em plataformas heterogêneas. Em cenários homogêneos, aplicamos as heurísticas Round Robin e Sorted-Greedy. Alcançamos um ganho de desempenho de 41% com a heurística Sorted-Greedy. No que diz respeito às soluções heterogêneas, implementamos as abordagens Area Bound Estimation (ABE) Guided e Heterogeneous Earliest Finish Time (HEFT), onde adaptamos ambas as soluções às nossas necessidades. Para a solução guiada por ABE, implementamos a programação linear em tal formato que também pudéssemos extrair o vetor de distribuição utilizado para alcançar o melhor makespan teórico. Também alteramos o algoritmo HEFT para considerar apenas o tempo gasto em cada ambiente heterogêneo para alocar novas tarefas a um trabalhador. Testamos essas duas estratégias em três cenários heterogêneos, onde obtivemos um ganho de desempenho de 78% em relação ao escalonador original.

Além disso, devido à característica estática de nossa carga, propomos uma ferramenta que auxilia no planejamento de capacidade para esta aplicação. Implementamos nossas quatro heurísticas de balanceamento de carga em um código baseado em R que simula, a partir de uma determinada entrada contendo o benchmark de qualquer caso de estudo e o número de máquinas envolvidas, um conjunto diversificado de combinações de máquinas possíveis e seu makespan para uma iteração do método. Nossa ferramenta oferece uma simulação precisa, o que nos permite realizar tal simulação antes de execuções reais da aplicação de inversão. Por esta simulação, podemos prever ainda o melhor cenário de execução com os recursos disponíveis que temos naquele momento.

Como trabalho futuro, pretendemos portar parte do código da aplicação para aceleradores de GPU. Uma vez que este método aplica muitas operações de matriz, podemos ganhar ainda mais desempenho na execução dessas operações em GPUs. Para fazer tal aprimoramento é necessário aplicar mudanças de código mais profundas na aplicação, que contam com um melhor entendimento dos métodos numéricos aplicados pelo processo de inversão.