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

FÉLIX DAL PONT MICHELS JÚNIOR

Optimization and adaptation of applications for vector processors

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

Advisor: Prof. Dr. Philippe Olivier Alexandre
Navaux

Porto Alegre
July 2022

CIP — CATALOGING-IN-PUBLICATION

Dal Pont Michels Júnior, Félix

Optimization and adaptation of applications for vector processors / Félix Dal Pont Michels Júnior. – Porto Alegre: PPGC da UFRGS, 2022.

81 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Philippe Olivier Alexandre Navaux.

1. Vector. 2. Optimization. 3. Performance Analysis. 4. High Performance Computing. I. Navaux, Philippe Olivier Alexandre II. Título.

ACKNOWLEDGMENT

I would like to thank my family, Elisabeth Dal Pont, Félix Sávio Michels and Paulo Henrique Dal Pont Michels, for all the material and emotional support during all my life.

My girlfriend and lifemate Mariana de Oliveira Barra Costa for its love and companionship, which without I would have not made it. To you, all the love I can give.

To my Advisor Prof. Dr. Philippe Olivier Alexandre Navaux, thank for your patience, guidance and all the opportunities.

To all the other contributors, small or large, direct or not, thank you.

ABSTRACT

Vector processors are designed to favor the execution of an instruction on multiple data, which increases the number of calculations per cycle, and subsequently improves performance in numerical applications, such as wave propagation and fluid mechanics.
In the same vein of performance improvements, quantum computing is becoming a reality with machines' latest announcements with nearly 100 qubits. In preparing the execution of quantum applications, simulators are used. One such simulator available to the public is Hiperwalk, a Quantum Walk simulator.
Furthermore, the importance of Controlled Source Electromagnetics (CSEM) has increased in the past decade. Along with this interest, its efficiency increased, data acquisition became more straightforward, and costs went down. For the Oil and Gas industry, modeling this data is necessary for exploration. The Modeling with Adaptively Refined Elements for 2D Electromagnetics (MARE2DEM), developed at Columbia University, is one of the tools used to model CSEM data.
Therefore, with performance as the main focus, this master dissertation will be divided into three parts. The first part is the performance analysis of two classic optimization techniques using two different applications. The second part is the adaptation of Hiperwalk and subsequent performance analysis. The third part is the adaptation and performance analysis of MARE2DEM. These applications will utilize the same target architecture to accelerate their execution, the NEC SX-Aurora TSUBASA, a vector processor. All three cases improved performance up to $1,9\times$, regarding the first part. The second part was able to increase up to 75% performance. Lastly, the third part reached a 27% improvement over the original implementation and architecture.

Keywords: Vector. Optimization. Performance Analysis. High Performance Computing.

Otimização e adaptação de aplicações em processadores vetoriais

RESUMO

Os processadores vetoriais são projetados para favorecer a execução de uma instrução em vários dados, o que aumenta o número de cálculos por ciclo e, posteriormente, melhora o desempenho em aplicações numéricas, como propagação de ondas e mecânica dos fluidos. Na mesma linha de melhorias de desempenho, a computação quântica está se tornando uma realidade com os últimos anúncios das máquinas com quase 100 qubits. Na preparação da execução de aplicações quânticas, são utilizados simuladores. Um desses simuladores disponíveis ao público é o Hiperwalk, um simulador de Quantum Walk.
Além disso, a importância da Eletromagnética de Fonte Controlada (CSEM) aumentou na última década. Junto com esse interesse, sua eficiência aumentou, a aquisição de dados tornou-se mais simples e os custos caíram. Para a indústria de Petróleo e Gás, a modelagem desses dados é necessária para a exploração. O Modeling with Adaptively Refined Elements for 2D Electromagnetics (MARE2DEM), desenvolvido na Columbia University, é uma das ferramentas utilizadas para modelar dados do CSEM.
Assim, tendo o desempenho como foco principal, esta dissertação de mestrado será dividida em três partes. A primeira parte é a análise de desempenho de duas técnicas clássicas de otimização usando duas aplicações diferentes. A segunda parte é a adaptação do Hiperwalk e posterior análise de desempenho. A terceira parte é a adaptação e análise de desempenho do MARE2DEM. Esses aplicativos utilizarão a mesma arquitetura de destino para acelerar sua execução, o NEC SX-Aurora TSUBASA, um processador vetorial. Todos os três casos melhoraram o desempenho em até $1,9\times$, em relação à primeira parte. A segunda parte conseguiu aumentar o desempenho em até 75%. Por fim, a terceira parte alcançou uma melhoria de 27% em relação à implementação e arquitetura originais.

Palavras-chave: Vetorial, Otimização, Análise de Desempenho, Computação de Alto Desempenho.

LIST OF ABBREVIATIONS AND ACRONYMS

CSEM	Controlled-Source Electromagnetic
CTQW	Continuous-Time Quantum Walk
DTQW	Discrete-Time Quantum Walk
EM	Electromagnetic
FEM	Finite Element Method
GPUs	Graphics Processing Units
HPC	High Performance Computing
LNCC	National Laboratory of Scientific Computing
MARE2DEM	Modeling with Adaptively Refined Elements for 2D Electromagnetics
MT	Magnetotelluric
NAS	NASA Advanced Supercomputing
PCAD	*Parque Computacional de Alto Desempenho*
QW	Quantum-Walk
RTM	Reverse Time Migration
SIMD	Single Instruction Multiple Data

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

High performance computing (HPC) is indispensable for many industries, commercial sectors and research today. It provides various benefits ranging from facilitating data analysis to enabling new simulations and modeling (EZELL; ATKINSON, 2016). Thus, advances in HPC are significant. By increasing the dimensions and decreasing the execution time, we can benefit different areas of society.

Application performance largely depends on the architecture used and how the program was coded to run on it. Currently, for acceleration purposes, different architectures are used, such as multicore, manycore, specific accelerators and GPUs (Graphics Processing Units).

A large number of architectures, while being attractive and flexible, imposes new challenges for the programmer (MITTAL; VETTER, 2015). The increasing complexity of the architecture will also increase the difficulty of implementing the application. It is worth mentioning the idea that there are several common bottlenecks, such as those found in the memory subsystem, which includes cache, thrashing pollution, among others.

In addition to using new architectures, several techniques are employed to boost performance, some of which are unique to specific architectures. For example, vectorization allows an instruction to act on multiple pieces of data, but not all of them implement support for it. Other optimizations include increasing the cache memory hit rate, which is possible on different architectures.

In this sense, NEC Corporation launched a new architecture, a vector processor called SX-Aurora. This processor has eight processing cores at 1.6 GHz and three levels of cache memory(KOMATSU et al., 2018). One of the advantages of this architecture in relation to the other existing ones is the size of its vector units, 256 units. Furthermore, the NEC compiler makes decisions automatically. That is, it identifies vectorizable areas and generates code for that. However, the compiler still needs help from the programmer to facilitate code interpretation, in addition to improving automatic vectorization, following specific guidelines and, in this case, using optimization techniques such as loop unrolling and inlining.

Vector architectures are Single Instruction Multiple Data (SIMD), which have great potential for highly parallelizable scientific applications. Among these are numerical applications, time prediction, multimedia processing (KSHEMKALYANI, 2012), collision simulation (HENNESSY; PATTERSON, 2019), data compression, and others. One prominent feature of these vector processors is the possibility of using an instruction to reproduce hundreds of operations. Moreover, all the results of the elements of a vector are independent, and therefore, checking the resulting data is not necessary. Memory access is done only once for each vector, inferring a small memory access latency.

1.1 Simulation

Another way to improve scientific output is through simulation. Simulators are necessary and powerful tools used in various fields, from research to commercial and industrial enterprises.

1.1.1 Oil and gas prospecting

The investments of the petrochemical industry in controlled-source electromagnetic (CSEM) have only increased in the last 15 years, and the academic interest goes along with it (CONSTABLE; SRNKA, 2007) (COOPER; MACGREGO, 2020). The main reason for these investments is to reduce risk. CSEM provides more data and, therefore, more information to give new insights into the seafloor bed, consequently reducing risk (FANAVOLL; GABRIELSEN; ELLINGSRUD, 2014).

CSEM data acquisition produces a large quantity of data, leading to a tremendous computational problem for solving inverse modeling. Some regions with complicated geometry may require additional data, mainly to produce full 3D inversion. However, 2D inversion is much quicker and provides a more straightforward interpretation of actual data in a shorter runtime, making it a more robust, sensible, and feasible approach (PRICE et al., 2008).

One necessary implementation to manage this kind of CSEM data is the application "Modeling with Adaptively Refined Elements for 2D Electromagnetics", referred to in this work as MARE2DEM (KEY, 2016). MARE2DEM is an open-source code for 2D inversion of CSEM data, magnetotelluric (MT) data, and surface-borehole EM data, by parallel adaptive finite elements for onshore, offshore, and downhole environments. Due to the large quantity of data provided by CSEM, high computational power is necessary to execute such data set with efficiency. Therefore, efficient code and a powerful computer are preferred.

1.1.2 Quantum Walk Simulators

Quantum computing advancements will inevitably increase computational power and efficiency (EASTTOM, 2021). These improvements are necessary as modern computers struggle to process complex biological systems, chemical structures and new materials (TRABESINGER, 2017). Even unlikely areas, like Virtual Reality, will be able to take advantage of this new technology (ZABLE et al., 2020). IBM has achieved 127 qubits in its latest quantum-computing chip, marking a new milestone for the whole industry, the first three-digit qubit count (BALL, 2021). IBM aims to surpass 1000 qubits by 2023, targeting a never seen increase in computational power.

However, these quantum computers and processors are not fully available to the general public. To fill this gap between the general scientific community and quantum computer science, quantum simulators are the best alternative.

A category of such simulators is quantum walk simulators, even though they are rare. One of those rare cases is the Hiperwalk simulator (LARA; LEãO; PORTUGAL, 2017). This simulator is divided into three parts. The first is a user interface written in Python that generates arrays and vectors based on user input in the main program. The second part consists of the calculations for the quantum steps. The Neblina-core library is used, developed at the National Laboratory of Scientific Computing (LNCC). This library facilitates matrix-vector and vector-vector type calculations in heterogeneous architectures. Finally, the last part is a module capable of calculating statistical distributions and generating output data files.

1.2 Objectives

This master's dissertation has performed an evaluation, adaptation and optimization of four separate applications in three distinct parts. These applications are the NAS benchmark, Reverse Time Migration (RTM), MARE2DEM and Hipewalk.

The first part mentioned above is to optimize the performance of real applications and a set of benchmarks using loop unrolling and inlining techniques, seeking to improve the automatic vectorization performed by the compiler. More precisely, the present work presents the following contributions:

- An experimental performance analysis of the NEC SX-Aurora TSUBASA vector accelerator is performed. A benchmark of parallel synthetic maps and an actual application is used.

- Show that the loop unrolling and inlining techniques are capable of significantly improving the performance of applications when executed in SX-Aurora TSUBASA in situations where the performance gain is not automatically by its compiler.

The second part is an investigation, adaptation and analysis of MARE2DEM's execution utilizing a vector architecture, the SX-Aurora, and the traditional x86 architecture. Two data sets will be used, a synthetic one and a real-world case provided by Petrobras, Brazil's oil & gas corporation. Therefore, the main goals of this research are:

- The implementation of MARE2DEM on NEC's SX-Aurora. The standard Intel math library was substituted for the MARED2DEM in the SX-Aurora because NEC's architecture does not support it. All entries that required Intel's math library were rewritten, supporting NEC's mathematical libraries.
- To investigate the performance analysis of this implementation by comparing it with x86 architecture CPUs.

The third part is regarding the adaptation and performance analysis of Hiperwalk in NEC's SX-Aurora. The main goals are:

- Introduction of the Hiperwalk simulator, explaining its use cases;
- The implementation of the kernel of HiperWalk on NEC's SX-Aurora TSUBASA;
- A performance analysis by comparing the NEC's HiperWalk adaptation with NVIDIA's GPUs.

The primary goal of this dissertation is to elevate the supposed potential that a vector architecture can have in the right circumstances, especially the SX-Aurora TSUBASA, e. g. high memory bandwidth.

1.3 Contributions of this work

The main contributions of this work are the performance analysis of classical optimization techniques for SX-Aurora using two different applications, and the code adaptation and performance analysis of a quantum walk simulator (Hiperwalk) and a geophysics fine element code used in the gas and petroleum industry (MARE2DEM). It was found that the optimization techniques were able to increase performance in most cases, and the code adaptations also had performance improvements under the right circumstances.

1.4 Document organization

This dissertation has been organized as follows. Section 2 presents the theory regarding the optimization techniques and different applications utilized in this work. Following, section 3 presents and discusses related work regarding vector architectures, MARE2DEM and Hiperwalk. The methodology, workflow and execution environment are presented in section 4. Section 5 depicts the experimental results of our two approaches, divided into three sections. Finally, section 6 presents the conclusion and possible future work.

2 BACKGROUND CONCEPTS

This chapter will provide brief explanations on four background concepts related to this dissertation and to its three parts, specifically the Quantum-Walk Models, optimization techniques, vectorization and CSEM.

2.1 Quantum-Walk Models

The first proposed quantum walk model was the Discrete-Time Quantum Walks (AHARONOV; DAVIDOVICH; ZAGURY, 1993). In 2003, Shenvi et al. proposed and developed a quantum search routine for a hypercube, achieving a time complexity of O($\sqrt[2]{N}$), where N is the number of vertices in the graph (SHENVI; KEMPE; WHALEY, 2003). Comparing it to the classical search algorithm, it has a quadratic gain. A similar method was developed by Ambainis et al. to create another quantum search algorithm for a two-dimensional lattice with a time complexity of O($\sqrt[2]{N.log(N)}$), close to being quadratically better than previous classic algorithms (O($N.log(N)$)) (AMBAINIS; KEMPE; RIVOSH, 2004).

Building upon Abainis et al. works, Tulsi introduces another qubit in the process, improving the time complexity (TULSI, 2008). There are even more improvements and different, however similar, methods in the literature, for example, Hein and Tanner (HEIN; TANNER, 2010), Abal et al. (ABAL et al., 2012), and Abal et al. (ABAL et al., 2010).

The notion of random quantum walks can be simplified to a spin -1/2 (spin is the angular momentum of elementary particles) particle going through motion in one dimension and its decision to change its motion taking a new step to the left or to the right. This decision is dependant on a calculation done within its z axis of its spin (AHARONOV; DAVIDOVICH; ZAGURY, 1993).

As a very through review is done Venegas-Andraca regarding quantum walks (VENEGAS-ANDRACA, 2012). In this work he touches and elaborates on many topics regarding the method. He gives the fundamentals of quantum walks, first starting with a base in random walks on an unrestricted line. Showing hitting time of classical and unrestricted classical discrete random walk on a line. Also, while on the fundamentals, he shows the Discrete Quantum Walk on a line, explaining the coin and its structure, the Schrodinger approach, the relation of Schrodinger and Discrete Path Integral Analysis to the Hadamard Walk.

Furthermore, Venegas-Andraca approaches with a general coin the Discrete Quantum Walk on a line and also shows it with boundaries. Now with several coins it is possible to learn about Unrestricted quantum walks on a line.

Most importantly its the Discrete quantum walks on graphs and the algorithms based on these quantum walks, especially concerning the Hiperwalk simulator. Venegas-Andraca gives more in depth knowledge about other subjects regarding quantum walks and also of great importance, touching upon quantum walks and its universality, the simulation of quantum walks and quantum algorithms utilizing classical computer paradigms and the simulation of quantum systems by quantum walks.

The most notable division in quantum walk models is the Discrete-time Quantum Walk (DTQW) and Continuos-time Quantum Walk (CTQW). Their differences are better described as "[...] DTQW is defined on a discrete-time domain, and in contrast, CTQW is defined on a continuous-time domain." (JAYAKODY; MEENA; PRADHAN, 2021).

Another well known quantum walk model is the Szegedy's Quantum Walk. Portugal et al. describes it perfectly: "Szegedy's QWs are obtained by quantizing classical discrete Markov chains described by some transition matrix. Szegedy also developed QW-based search algorithms inspired by a previous known coined-based search algorithm. "

2.2 Optimization techniques

In Part one of this work it is implemented two distinct techniques that are widespread, inlining and loop unrolling.

Inlining is the technique of placing a copy of a function when it is called. This improves execution by eliminating unnecessary overheads such as function call instructions related to using registers and stacks. However, if the function to be applied to the technique is very large, that is, it takes up much space, inlining can end up consuming a lot of the instruction cache memory, deteriorating the application's performance. Thus, it is recommended to use it for functions that take up little space.

Due to the nature of inlining the function will be expanded inside of another function. This will increase its spatial locality as well the number of function calls, which may in turn decrease the cache miss ratio (CHEN et al., 1993). With the enlargement of the code lines, the compiler can more effectively place its instructions, even improving load forwarding performance due to sequentially growth (CHEN et al., 1993).

The loop unrolling technique aims to transform a loop so that it applies more than one execution per iteration. This minimizes the number of hops and instructions. In addition, it can facilitate parallelization and vectorization, as long as the loop variables are independent.

So, loop unrolling generally becomes effective when the compiler can simulate the computations done by the loop control (HUANG; LENG, 1999). However, with the help of the programmer, before the compiler even touches the code, it is possible to efficiently insert this technique, even in sections that the compilers can't recognize.

2.3 Vector processors and Vectorization

As defined by Hennessy:

> Vector architectures grab sets of data elements scattered about memory, place them into large, sequential register files, operate on data in those register files, and then disperse the results back into memory. A single instruction operates on vectors of data, which results in dozens of register–register operations on independent data elements. These large register files act as compiler-controlled buffers, both to hide memory latency and to leverage memory bandwidth. Since vector loads and stores are deeply pipelined, the program pays the long memory latency only once per vector load or store versus once per element, thus amortizing the latency over, say, 64 elements. Indeed, vector programs strive to keep memory busy. (HENNESSY; PATTERSON, 2012, p. 264)

One of the main features of these vector processors is the possibility of using one instruction to reproduce hundreds of operations. In addition, all the results of the elements of a vector are independent of each other, and therefore, it is not necessary to check the data for risk. Memory access is done only once for each vector, inferring a small memory access latency.

Of of the main factors today for code vectorization and its success is the structure of these applications. True data dependencies, which are inherit to some programs, are to be avoided and, if possible, removed entirely (HENNESSY; PATTERSON, 2012).

In the case of NEC's latest vector machine, the compiler handles automatically the vectorization where it is possible to apply. However the programmer should assist the compiler in recognizing the sections of code to be vectorized. To achieve that, NEC provides a most useful manual (NEC, 2020a).

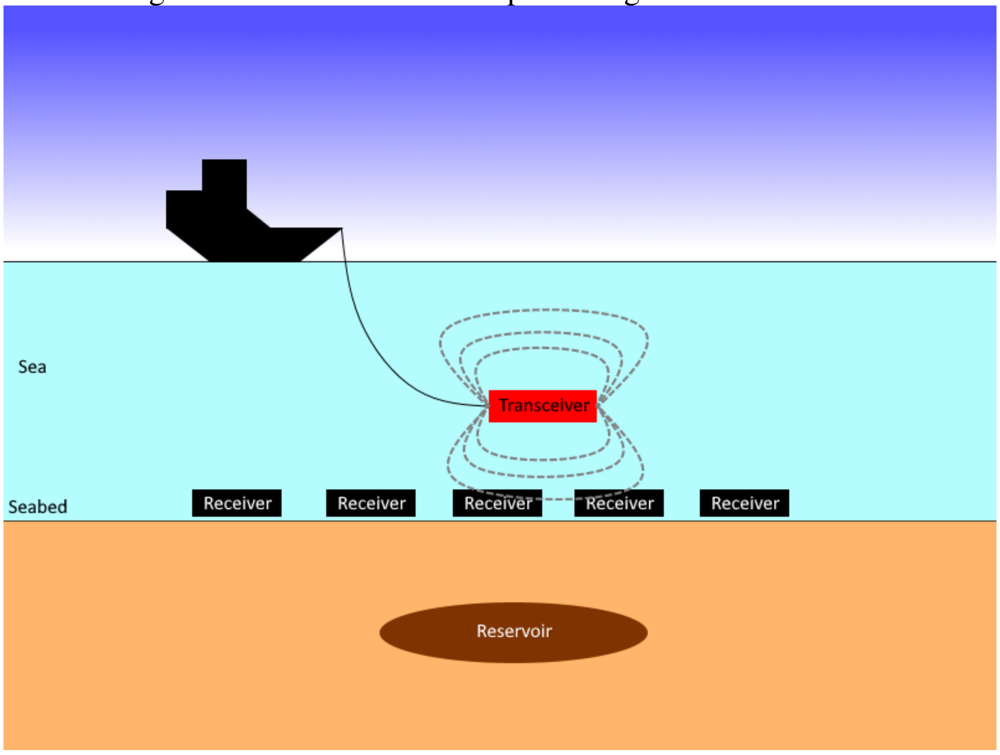
Moreover, SX-Aurora main selling point is its peak performance of 3,07 TeraFLOPs and the vector length of 256 units. These two features are extremely important to keep in mind. Due to the nature of vector processors, vector length is self explanatory. Much of the performance can be associated to it, as more calculations are done per instruction, higher performance is achieved. Whoever, memory bandwidth is different.

This advantage is primordial to some programs and applications that deal with higher quantities of data.

2.4 Controlled-source electromagnetic method

Controlled Source Electromagnetics (CSEM) is a marine geophysical method mapping the subsurface resistivity. The CSEM method works as follows: A electrical-field transmitter is rested close to the seafloor but not at the bottom. Then, electromagnetic receivers are spaced at specific ranges on the seafloor, resulting in the CSEM data in these range intervals. Figure 2.1 represents the process mentioned above.

Figure 2.1: Schematic view representing the CSEM method.



So, a boat tows a horizontal electric dipole source (HED), the transmitters, across a line (2D) or grid (3D) of receivers. Through the emission of and electromagnetic field the signal is able to propagate, traveling in the subsurface and back to the receivers.

A complete evaluation of CSEM is described by Chave et al. " Controlled source EM methods utilize time-varying electric and magnetic dipole sources of known geometry to induce electric currents inside the conducting earth " (CHAVE; CONSTABLE; EDWARDS, 2012).

Also, CSEM adheres, as other electrical methods, to the equivalency principle which states, as explained by Mehta et al., " the response depends, within limits, primarily on the transverse resistance of the target (resistivity thickness product) ". This then excludes that the response depends individually on the thickness and resistivity.

3 RELATED WORK

This chapter will present several scientific articles and manuals from the areas explored in this dissertation. Thus, studies in the fields of parallel and high performance computing (general optimization techniques and SX-Aurora), quantum computing (Quantum walk theory and simulation), geophysics (MARE2DEM and CSEM) and instrumentation (Score-P).

The general structure will be a summary followed by a brief discussion around its relation to this dissertation.

3.1 SX-Aurora

The work by Komatsu et al. (KOMATSU et al., 2018) highlights the potential of the SX-Aurora TSUBASA architecture, comparing it to other architectures, including Intel Xeon Phi 7290, NVIDIA Tesla V100 and SX-Ace. This comparison is made by running benchmark applications and two real applications. The results show that the SX-Aurora architecture can run efficiently, up to $3.5\times$, in addition to getting a higher speedup of up to $2.8\times$.

Yokokawa et al. (YOKOKAWA et al., 2020), demonstrates the capability of the SX-Aurora TSUBASA architecture for I/O applications, comparing the I/O system of the new architecture with typical I/O systems. The distinct accelerated I/O function present in the architecture, for some instances, triples the performance in MB per second.

A performance evaluation of SX-Aurora is done by Komatsu and Kobayashi (KOMATSU; KOBAYASHI, 2020). It aims to clarify its potential against benchmark programs. It uses Stream benchmark to evaluate the memory bandwidth, then the Himeno and the HPCG benchmarks. They conclude that from these benchmarks, the remarks concerning the high memory bandwidth can be sustained, achieving from 1.37 to 11.7 times more bandwidth in the Stream benchmark, compared with 4 other architectures, including its predecessor, SX-Ace.

In a cross between this chapter and the quantum chapter, Kobayashi and Komatsu report an overview of an ongoing project concerning a "Quantum-Annealing Assisted Next Generation HPC Infrastructure and its Applications" (KOBAYASHI; KOMATSU, 2021). This infrastructure combines HPC and Quantum-Annealing engines. SX-Aurora takes a center stage in this infrastructure, therefore, a performance evaluation is also presented. Throughout their analysis, they show the potential of combining SX-Aurora and Quantum Annealing to provide high-performance and high-sustained computing, through SX-Aurora capacity to increase performance of memory intensive programs and the Quantum Annealing possibilities in data clustering optimization.

3.2 Optimization Techniques

It is proposed by Serpa et al. (SERPA et al., 2017) different optimization strategies for a wave propagation model in different architectures. The study shows that using techniques such as loop interchange, vectorization and mapping of data and threads, it was possible to achieve a performance increase of up to 8.5 times.

Memory alignment and cache blocking techniques are presented by Castro et al. (CASTRO et al., 2016), improving the performance of an acoustic wave propagation application. Similar to Castro et al., in order to achieve better performance, in this work, two techniques will be implemented, loop unrolling and inlining.

An overview of optimization techniques for cache memory is provided by Kowarschik and Weiß (KOWARSCHIK; WEISS, 2003), demonstrating techniques such as: loop interchange, loop fusion, loop blocking, among others. Some of these techniques are developed in the present work, observing their effects on a vector architecture.

Jacquelin et al. (JACQUELIN; MARCHAL; ROBERT, 2009) propose algorithms that use cache memory for multicore architecture and that minimize memory access time, thus decreasing cache memory failure rate, as well as timely access to shared memory and cache. The impact of cache memory failure rate is also explored in this article, focusing now on a vector architecture.

In the work by Sherlon et al. (SILVA; SERPA; SCHEPKE, 2016) is compared different optimization techniques, such as loop unrolling and loop tiling. It demonstrates its impact on parallel applications, achieving performances of up to 20 times the original. In this work, we intend to analyze the impact of loop unrolling and inlining in a vector architecture, different from the focus on the Xeon processor of (SILVA; SERPA; SCHEPKE, 2016).

Theodoris et al. work shows that predicting optimal usage of inlining optimization is non-trivial and a ongoing effort to improve compilers decision making (THEODORIDIS; GROSSER; SU, 2022). Mainly, the paper shows an investigation of optimal inlining utilization by applying SPEC2017 benchmark. The result of their work is shown by the creation of an efficient autotuning strategy for the inlining technique, increasing its performance by up to 27%, in the case of the SPEC2017 benchmark.

Rodrigo et al. proposes in their work a novel loop rolling technique called RoLaG (ROCHA et al., 2022). This new technique is capable of detecting isomorphic code through SSA graphs. To compare, SPEC2017 and AnghaBench were utilized to benchmark this new technique. They show it was possible to achieve up to 2.7% and 9.12% code reduction, respectively.

3.3 MARE2DEM

Myer, through the use of MARE2DEM, explores 2D inversion of marine CSEM and MT data (MYER; KEY; CONSTABLE, 2015). The primary purpose is to make the inversion aware that the subsurface comprises similar geologic domains. The presented workflow is subjected to CSEM limitations, making fine-scale structural details hard to be resolved. However, in confounding settings, it is still possible to be used to map the rough qualities and general extent.

Grayver presents it as a new 3-D parallel inversion scheme for CSEM data in the frequency domain based on a direct forward solver (GRAYVER; STREICH; RITTER, 2013). Gauss-Newton minimization provides data of model inversion, proving the approach's applicability to real-world problems, showing real-world data sets being possible to manage with only medium-sized clusters.

The application MARE2DEM (KEY, 2016) implements 2D forward and inversion modeling algorithms for MT and CSEM data. The inversion modeling uses CSEM information provided by a grid of electromagnetic receivers resting on the seafloor bed and its response to a transmitter below the sea's surface while being towed by a ship. This response happens in known intervals and later is converted into electrical resistivity, making it possible to study the components below the seabed.

Furthermore, MARE2DEM employs finite element method (KEY; OVALL, 2011) to find the resistivity model. To accurately calculate CSEM and MT models' solutions, Kerry Key presents a parallel goal-oriented adaptive finite element method in his work. He also introduces a reliable goal-oriented error to guide the iterative mesh refinement. The overall performance is assessed utilizing clusters of 800 processors with real-world data sets, achieving execution times of only a few seconds.

Yavich and Zhdanov published their work on improving the finite element modeling through finite difference (YAVICH; ZHDANOV, 2020). This novel implementation used a finite difference solver based on implicit factorization of the matrix. A comparison is made between their implementation and three other public available programs. One of them is MARE2DEM, and it was possible to achieve improved performance with similar accuracy.

Score-P is a unified performance-measurement infrastructure used for profiling, event tracing, and evaluating HPC applications (KNÜPFER et al., 2012). It provides a variety of different performance evaluation tools, such as Tau (SHENDE; MALONY, 2006), Periscope (GERNDT; FÜRLINGER; KEREKU, 2005), Scalasca (GEIMER et al., 2010), and Vampir (KNÜPFER et al., 2008). Each one addresses Score-P behavior, utilizing certain factors. Score-P is a complete tool that focuses on bringing the tools mentioned above and being the central piece that binds them together, accommodating all components.

Hanbo et al. shows in his article a novel wavelet Galerkin method (CHEN; XIONG; HAN, 2022). This new method is important, mainly, to solve the forward modeling problem present in Marine Controlled-Source Electromagnetic Method (MCSEM). To evaluate this new method, it was compared to the finite element and difference and the analytical methods. Their comparison shows that this new method has an advantage in memory utilization and computing time.

3.4 Hiperwalk

The theory of quantum walks is constantly evolving. The work of Willsch et al. demonstrates that quantum walks can be modeled without using the notion of particle-wave duality. For the simulation of quantum walks, the simulator reproduces the experimental data of an application of quantum walks through atoms in a given trajectory, starting from the concept of refutability. Any classical mechanics model can be excluded to calculate the trajectory of the atom (WILLSCH et al., 2020).

Panahiyan applies quantum walks similarly to Pedro Lara (LARA; LEãO; PORTUGAL, 2017). Using a step-dependent simulation using the coin concept, it simulates topological phases and invariants, boundary states, and the possibility of phase transition (PANAHIYAN; FRITZSCHE, 2019).

Matsuura (MATSUURA, 2021) describes the promise and challenges of bringing quantum computing out of the lab and into a commercial complete computer system and will give an overview of Intel's quantum computing research and system development. Many interdisciplinary research questions cross the boundaries between physics, engineering, and computer architecture in constructing a full-stack quantum computing system.

The paper Simulation of Quantum Walks using HPC by Pedro Lara, Aaron Leão, and Renato Portugal describes the Hiperwalk simulator used in this work (LARA; LEãO; PORTUGAL, 2017). This work is one of the few quantum walk simulators available to the public and one of the only ones to use parallelism for the simulation. This simulator works with vector architectures such as SX-Aurora, as well as other SIMD architectures, like NVIDIA GPUs. However, the simulator is based on languages and libraries that restrict portability, such as OpenCL, thus being a good source for adaptation.

The QWalk simulator (MARQUEZINO; PORTUGAL, 2008) is a C-program for simulating quantum walks on lattices. The pyCTQW simulator (IZAAC; WANG, 2015) aims to simulate multi-particle continuous-time quantum walks using distributed memory. The QSWalk.jl simulator (GLOS; MISZCZAK; OSTASZEWSKI, 2019) is a Julia package that aims to simulate quantum stochastic walks on directed weighted graphs.

Anglés and Pérez explore in their article a quantum walk that is capable of simulating the pattern of a 1/2 particle spin with an extra dimension in a ordinary spatial dimension with warped geometry (ANGLéS-CASTILLO; PéREZ, 2021). Through the Dirac equation it is possible to foresee properties of the quantum walk. As the result, they are able to correlate localization and high energy physics in quantum walk, finally being possible to conclude that for simulation of field theory models with more dimensions reliant on curvature of space-time, quantum walks are prime contenders.

3.5 Final Thoughts

When comparing the works presented previously and the present work, the focus on the new SX-Aurora TSUBASA architecture is a differential due to its essence of vector processor and its unique compiler, which, for example, promotes automatic vectorization. Another great feature of NEC's vector architecture is its very large 256 units vector length, which contributes to the processors ability to manipulate matrices. Also, it is imperative to note the other main selling point of SX-Aurora, its high bandwidth memory. To achieve high performances, this advantage must be utilized, which implies tailoring the applications and choosing the right ones initially, to best harvest this feature.

Furthermore, even in the works presented on the new architecture, none of them implements simple and known optimization techniques, which is the case of those applied in this work, the loop unrolling and inlining techniques.

4 METHODOLOGY

This chapter will focus on all three parts' general and specific methodologies and the instrumentation to acquire data. First, the execution environment and the standard methodology for experimentation and testing. Next, the specifics for each part and the changes implemented for adaptation. At last, the instrumentation with perf, FTRACE and score-P.

4.1 General Methodology and Infrastructure

The general experimentation workflow follows four steps: preparing the environment, running the experiments, collecting data and analyzing. Figure 4.1 shows the steps with additional information. The blue-green rectangles in the middle represent the significant steps. The shapes with red color are the three experiments performed. Moreover, the green circles show additional work related to that step.

The execution environment preparation means creating the routines for compilation/execution and data visualization. Secondly, the experiments selected in the previous step are carried out in the work environment. The third workflow step is data collection and management, using R, score-P, perf or FTRACE.

4.1.1 Infrastructure

The experiments utilized the resources of the *Parque Computacional de Alto Desempenho* (PCAD) infrastructure, <http://gppd-hpc.inf.ufrgs.br>, at INF/UFRGS. Figure 4.2 shows a detailed schematic of the SX-Aurora architecture. The environment has eight-core, global memory, and cache L3, each core with memory cache L1 and L2, one unit of scalar processing (SPU), and a vector processing unit (VPU), with each VPU containing load buffer, store buffer, and 32 vector parallel pipeline (VPP) (NEC, 2020d). The table 4.1 shows the detailed specs of the architecture, containing the specs of the processors, cache, and global memories.

The Intel Cascade Lake microarchitecture is utilized in part two of this work, involving MARE2DEM, referred there as the x86 architecture. In Table 4.2 you have the specifications of the Intel Xeon Gold 6226 processor, which has 12 cores operating

Figure 4.1: Workflow of this dissertation general steps.

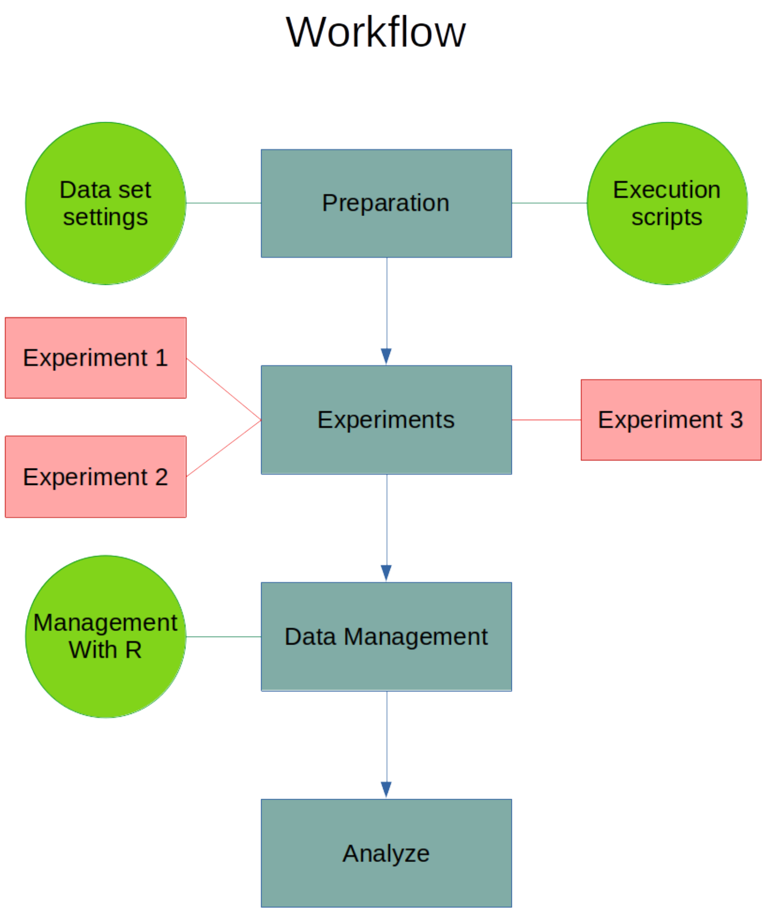
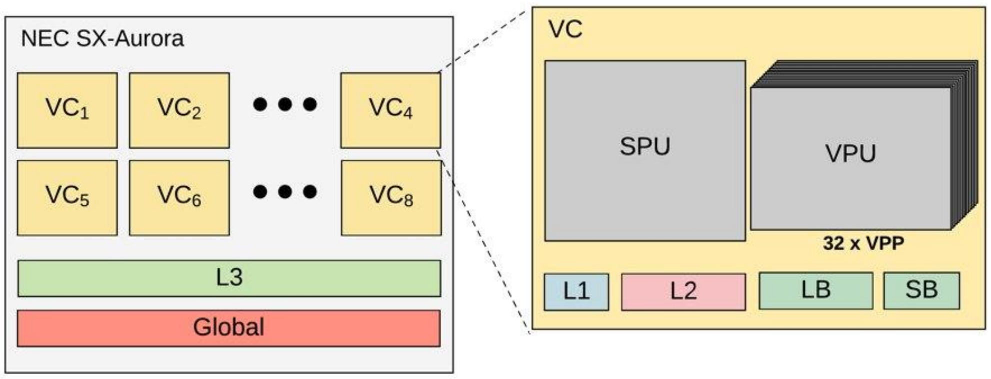


Figure 4.2: Detailed scheme of each SX-Aurora core.



at a frequency between 2.7 GHz and 3.7 GHz. Each core has 32 KB of the L1 cache of data and instructions and a private 1 MB L2 cache. The L3 shared across all cores has a capacity of 16.5 MB, and the machine also features 192 GB of DRAM memory (PEREZ et al., 2018).

For the tests in the third part, regarding Hiperwalk, we used a Tesla machine. It consists of 2 x Intel Xeon E5-2699 v4 Broadwell (Q1'16), 2.2 GHz, 44 cores (22 per

Table 4.1: SX-Aurora Architecture.

Vector Engine	Type 10BE
Processor	8 cores @ 1408 MHz
Microarchitecture	SX-Aurora
Cache	8 X 32 KB L1I 8 X 32 KB L1D;
	8 X 256 KB L2; 8 X 2 MB L3
Memory	HBM2 48 GB, 900 MHz

Table 4.2: Microarchitecture Cascade Lake (x86).

Processor	2 X 12 cores @ 2700 - 3700 MHz;
Microarchitecture	Cascade Lake
Cache	12 X 32 KB L1I; 12 X 32 KB L1D;
	12 X 1 MB L2; 16,5 MB L3
Memory	DDR4 192 GB, 2933 MHz

CPU), totaling 44 cores and 88 threads, 256 GB DDR4 RAM, 4 x NVIDIA Tesla P100, Pascal, 4 x 3584 CUDA Threads.

The Intel Cascade Lake microarchitecture represents the x86 architecture with NVIDIA's P100 GPU. In Table 4.3 you have the specifications of the Intel Xeon Gold 6226 processor, which has 22 cores operating at a frequency between 2.7 GHz and 3.7 GHz. Each core has 32 KB of the L1 cache of data and instructions and a private 1 MB L2 cache. The L3 shared across all cores has a capacity of 16.5 MB, and the machine also features 192 GB of DRAM memory. (PEREZ et al., 2018). The NVIDIA P100 is a Pascal GPU with 3584 CUDA cores (NVIDIA, 2016).

4.2 NAS benchmark and RTM

For the evaluation of SX-Aurora, we used the benchmark NAS (BAILEY et al., 1991) and an actual application used by the oil industry for seismic migration, called Reverse Time Migration (RTM) (ZHOU et al., 2018; FOWLER; DU; FLETCHER, 2010; FLETCHER; DU; FOWLER, 2009). The NAS Benchmark is a series of programs explicitly designed to evaluate the performance of parallel computers. The RTM modeling constitutes the simulation of the propagation of waves through time, using the acoustic

Table 4.3: Microarchitecture Broadwell (x86).

Processor	22 cores @ 2200 MHz;
Microarchitecture	Broadwell
Cache	12 X 32 KB L1I; 12 X 32 KB L1D;
	12 X 1 MB L2; 16,5 MB L3
Memory	DDR4 256 GB, 2933 MHz
GPU	NVIDIA P100, Pascal, 3584 CUDA

equations and the fact that different geological layers have different speeds.

Thus, the applications used for the benchmark NAS were: BT, CG, EP, FT, IS, MG, SP and UA. The input data class used is B. Specifications for these programs are found in the NAS documentation (BAILEY et al., 1991). All experiments, benchmark NAS and RTM modeling, were run ten times, then taking the mean and standard error.

Figure 4.3: Fortran source code snippet, illustrating a loop unrolling applied to the benchmark NAS.

```
do k=ksize-1,0,-1
   do m=1,BLOCK_SIZE
      do n=1,BLOCK_SIZE, 2
         rhs(m,i,j,k) = rhs(m,i,j,k)
            lhs(m,n,cc,k)*rhs(n,i,j,k+1)
         rhs(m,i,j,k) = rhs(m,i,j,k)
            lhs(m,n+1,cc,k)*rhs(n+1,i,j,k+1)
      enddo
   enddo
enddo
```

Both applications will receive the same optimizations. The first technique is inlining which consists of replacing the function call with the function itself. In this work, the inlining technique was made by combining two modes, using the flag `-finline-functions` and manually, and thus, the technique can be applied to functions specific, exploring only the benefits of the technique. In addition, the loop unrolling technique was also applied, which is the action of unrolling the loop. In the following sections, we have examples of the loop unrolling technique, specifically the Algorithm 4.3 and Algorithm 4.4, which show the technique applied in the benchmark NAS and in RTM modeling.

The version used is NPB 3.4.1 OpenMP (BAILEY et al., 1991). The applications BT, CG, EP, FT, IS, MG, SP and UA were executed in their original versions. The BT program is a tri-diagonal block solver. CG consists of the conjugate gradient method with irregular memory and communication access. EP is an Embarrassingly parallel application. FT is the fast Fourier transform in 3D. Integer sorting is implemented in the IS program. MG is a 3D potential field solver. SP is similar to BT, being a pentadiagonal scalar solver. Finally, we have UA, a solver for an unstructured adaptive mesh.

The compilation uses the exclusive NEC compiler and some flags of this compiler, namely `-O2` and `-fopenmp` (NEC, 2020b). It is referred to here as the "original" version the unmodified NAS code only applied to the automatic vectorization of the SX-Aurora TSUBASA machine. The “optimized version” consists of applying the optimizations presented above, inlining and loop unrolling.

Figure 4.4: Snippet of C source code, illustrating an unrolling loop applied to RTM

```
for (i=1; i<sx*sy*sz; i+=4) {
  maxvel=fmaxf(maxvel,vpz[i]*sqrt(1.0+2*epsilon[i]));
  maxvel=fmaxf(maxvel,vpz[i+1]*sqrt(1.0+2*epsilon[i+1]));
  maxvel=fmaxf(maxvel,vpz[i+2]*sqrt(1.0+2*epsilon[i+2]));
  maxvel=fmaxf(maxvel,vpz[i+3]*sqrt(1.0+2*epsilon[i+3]));
}
```

Data collection is done through NEC's profiling program, PROGINF and FTRACE (NEC, 2020c). This program gives us much information regarding the architecture, and for this study, mainly the FLOPS and the cache memory failure rate.

The compiler, when inserting specific flags, in this case `-report-all`, `-fdiag-inline=2`, `-fdiag-parallel=2` and `-fdiag -vector=2` (NEC, 2020b), makes a complete report, containing, for example, vectorized and non-vectored loops, functions that block vectorization, nested loops, extended loops, among others. This helps in deciding where to apply the optimizations. For example, for the BT program, the compiler was not vectoring some of the areas of computation, specifically the subtraction and multiplication parts of matrices. Analyzing the code and using the guide developed by NEC (NEC, 2020b), it is possible to apply the optimization techniques already discussed and guidelines to facilitate vectorization by the compiler. The Algorithm illustrated in Figure 4.3 shows the application of the loop unrolling technique in the BT program.

4.3 MARE2DEM

In part two, three experiments are presented, two of them utilizing an artificial demonstration data set and another one using a real-world data set provided by Petrobras. The artificial demonstration data set is relatively small compared to the real-world data set, with only 1028 and 17054 data points. It serves to test the application, providing validation to any modification implemented. Petrobras provided the second data set. The second data set is real-world data collected on the ocean floor utilizing the proper set of equipment. Since it has large amounts of data, it is perfect for testing the SX-Aurora memory bandwidth advantage.

The compiler flags for MPI Fortran (mpif90 v.7.5.0), MPI C (mpicc v.7.5.0), NEC MPI Fortran (mpincc v.2.13) and NEC MPI C (mpinfort v.2.13) utilized were `-O2`, `-fPIC`, `-fpp` and `-cxxlib`. These last two are exclusive to MPI Fortran.

The experiments selected in the previous step are carried out in the work environment. As mentioned, there are three experiments.

The first one uses the artificial demonstration data set. Several workers and data groups need to be chosen to run the application. For the x86 architecture, it was utilized 24 workers corresponding to the 24 cores of the CPU and the default group distribution. NEC's architecture utilized eight workers and the same default distribution. The second experiment also utilizes the artificial demonstration data set. However, instead of 24 workers for x86 architecture and eight workers for SX-Aurora, only two will be used, which will be better explained in Section 5.2.3. Last, the third experiment uses Petrobra's data set. It utilizes 24 and 8 workers for the x86 architecture and the SX-Aurora architecture. Ten executions were performed for all experiments to improve statistical rigor, achieving the mean execution time and error.

The main reason to utilize only one iteration for each experiment is due to the amount of time that one whole execution of the entire data, until its stop parameter, is around 15 to 20 days. So executing these three experiments, 10 times each to ensure there are no outlier data, without counting failed attempts and other incidents that might occur in 20 days of computing time, it would be necessary at least 450 days. So to achieve a faster preliminary, whoever robust and accurate result, we needed to execute only one iteration. Whoever to achieve the final result for the visual outcome validation we indeed run the program to its conclusion.

MARE2DEM implements a math library exclusive to Intel's processors. To execute MARE2DEM in the vector architecture SX-Aurora, this math library needs to be substituted by a general math library or implement NEC's math library, which works specifically to the needs of its vector engine. It was concluded that the latter is more desirable for the present work. It provides better support for parallelization and vectorization, and the nature of the library origin, being the same manufacturer of the vector engine, facilitated the porting process. Besides the math library, there were several other bugs and code rearrangements that needed to be performed.

Curiously, there were minor discrepancies in the porting process due to different FORTRAN versions. Some syntaxes that are utilized in FORTRAN nowadays are not supported. Also, some code structures are very unreliable to work with. One of them is the bind(C) interface, making FORTRAN's subroutines and functions compatible with C code. This section did not work at all, and luckily it was possible to rearrange the code not to need it.

One example of a substituted function is for sparse linear algebra equations. The original implementation relied heavily on intel's libraries, which were changed to use mostly NEC's functions. Using NEC's math library, it was possible to use its Sparse Basic Linear Algebra Subprograms (SBLAS) and HeteroSolver routines. The basic structure of this function follows: matrix storage, handle initialization, solution and finalization. The following paragraphs explain these stages.

In this implementation, we utilized Compressed Sparse Row (CSR) format. It consists of representing a matrix through three vectors. So the first step consists in storing the matrix in these three arrays. Nonzero entries are stored in the first vector. The column indices of these nonzero elements are kept in the second array. The third array stores the cumulative sum of the row counts.

Secondly, we set up the handles, which are all provided by the library. We need to call for the function that creates the handles while passing all the necessary parameters, for example, the CSR arrays and column and row counts.

Next, the math library provides the solution through the sblas_execute_mv_rd function. We need to provide the input we already created in the previous steps. Lastly, we need to finalize our code by returning our solution vector and destroying the handle.

In terms of mathematical functions, six needed to be replaced or recreated. There are two functions for factoring a sparse matrix, one for complex numbers and another for real numbers. Both needed to be recreated entirely, as the workflow of both libraries is very different. The same goes for the sparse linear algebra solver, one function each for complex and real numbers. Both were recreated, as described above. Lastly, two functions to free memory for both were replaced.

A validation of the adaptations to the math library of MARE2DEM was performed. This is necessary to assert that all the modifications made to the math library are not impacting in the final outcome of this application. Furthermore, in MARE2DEM, a visual validation is also necessary, due to the changes not only on the math library, but also in all code. This visual validation is shown in the investigation and results, chapter 5.

To achieve this validation simple mathematical problems were run with both the original and the modified libraries. Then the results were compared via subtraction. If the final result is zero, the modification is validated mathematically. The same matrix and vector input was utilized in all tests. All test came back zero, so all modifications were validated.

4.4 Hiperwalk

Hiperwalk is an open-source program to generate the dynamics of known quantum walk (QW) models in a generic graph using HPC. There are four different models: Discrete-Time Quantum Walk, Szegedy's Quantum Walk, Continuous-Time Quantum Walk , and Staggered Quantum-Walk.

In this version of the Hiperwalk simulator, only two of these models are presently implemented: the DTQW model and the Staggered Quantum-Walk. As this work utilizes the DTQW model, we will explain it and show its evolution. Furthermore, we will be elucidating how Hiperwalk process this model and show an example output. Plans for the simulator include implementing the remaining two quantum walk models.

The quantum walk simulation in Hiperwalk has three most significant steps: 1) Input parameters conversion, which includes underlying structures (line and cycles), coin operators, initial stats, and others; 2) Utilization of converted parameters in a quantum state, generating a vector in an extensive range of values; and 3) Interpretation of said vector through statistical analysis (probability distribution).

To achieve its goal, Hiperwalk utilizes high-level abstraction. It starts with a simple text file as input. The Neblina-core math library handles all calculations. We first choose our quantum walk model inside the input text file. Then the number of steps we will be taking. The input graph, with type and size. Finally, we need the initial state of the quantum walk. There are more optional commands, and some are exclusive to each quantum walk model. Figure 4.5 shows us an example output of the Hiperwalk simulator.

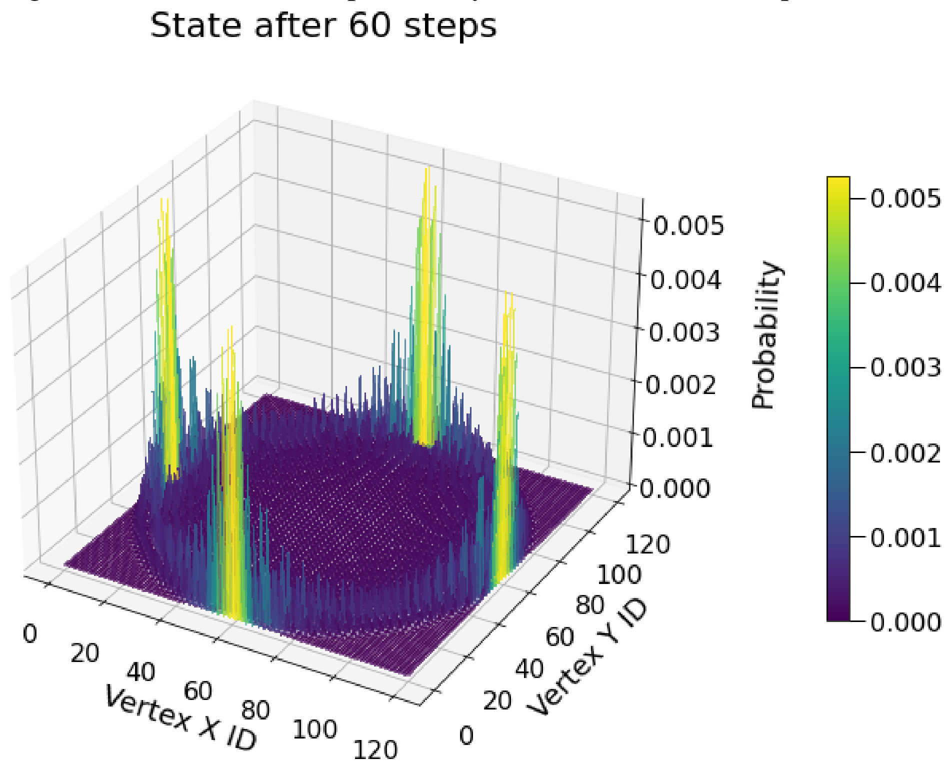
In Figure 4.5 we have the probability distribution of a quantum walk in a 121 x 121 mesh after 60 steps. The x and y-axis give their positions, and the z-axis provides the probability [%].

4.4.1 Neblina

Neblina-core is a math library that requires minimal knowledge about parallel programming from the user by establishing an easy-to-use parallel computing layer. Programming in Neblina-core is done sequentially. The interpreter sends data to the CPU or GPU through a parallel OpenCL API, independently of the processor's architecture, platform, or vendor.

However, as stated before, Neblina-core is implemented using a coding language

Figure 4.5: Quantum walk probability distribution in a 121 square mesh.



not compatible with NEC's architecture. To execute HiperWalk in the vector architecture SX-Aurora, all code sections that use OpenCL need to be substituted by implementing NEC's math library.

Similar to MARE2DEM, it was necessary to validate the adaptations to the math library of neblina-core that were implemented. Again, this is necessary to be sure that the final outcome does not suffer any influence due to the modifications made to neblina-core, Hiperwalk math‘s library.

Utilizing the same approach of MARE2DEM, however larger in scope due to the amount of modified functions, we utilized simple mathematical problems with both the modified and the original neblina-core. Then the results were compared via subtraction and if the final result was zero, it meant that the modifications made were mathematically validated. As before, to ensure this, it was utilized the same matrix and vector input in all tests. In neblina-core all modifications were validated through this system, as there were no further modifications done to the main Hiperwalk code, only to neblina-core.

4.4.2 Experiments

The execution environment creates the routines for compilation/execution and data visualization. Four data sets are presented in this work, gradually increasing in size. These data sets are square matrixes with x amount of nonzero elements. Each matrix has 1024, 4096, 16384, and 32768 nonzero entities. Since the last data set has large amounts of data, it is perfect for testing the SX-Aurora memory bandwidth advantage over NVIDIA's.

Because the neblina-core adaptation made for SX-Aurora uses the same functions as the original implementation, we can use the same Hiperwalk code with minor changes.

Each dataset was executed 20 times, and the average of each result is presented in the results section. The resulting data recorded was the runtime in seconds, MFLOPs, and cache hit ratio for L1 and LLC.

Inside Hiperwalk, we have different configurations we need to set. First, we need to choose how many calculations repetitions will be performed. For all tests, the same amount of 1000 repetitions was chosen. Second, we can choose which quantum walk model we want to use, in our case, DTQW.

Instructions for installing the simulator are available at the link: <http://qubit.lncc.br/qwalk/>. Neblina-core can be found at: <https://paulomotta.pro.br/wp/2021/05/01/pyneblina-and-neblina-core/>.

4.5 Data collection and Hardware Counters

The use of hardware counters, structures found in many modern processors and accelerators that allow the monitoring of events internal to these architectures. Some of these events are the number of instructions executed, the number of cycles, and the number of memory accesses.

Using these counters makes it possible to collect information in a more specific and detailed way than information obtained with other higher-level tools.

The user can identify the available information for his specific architecture and combine different counters to investigate different aspects. In addition, counters from different cores can also be combined, analyzing the system as a whole.

Thus, using hardware counters to analyze the performance of parallel applications allows the user to have more control over the process, with less noise on the application.

4.5.1 Linux perf

Linux perf is a profiling tool for Linux systems that allows access to the `perf_events` interface of performance events present in superscalar architectures. The tool is used on the command line using the `perf` command, and it is possible to list the events available on the platform using the `perf list` command. A command commonly used for performance analysis is the command `stat`, which allows the collection of information regarding the execution of an application. Using the `-e` option, it is possible to list the events to be collected. The example below represents the command line needed to collect information about the number of instructions and the number of CPU cycles required to execute the command `ls`:

```
perf start -e instructions,cpu-cycles ls
```

In addition to executing the command `ls` and listing the contents of the current directory, executing the command `perf stat` with the events `instructions` and `cpu-cycles` gives a similar result as noted below:

```
 Performance counter stats for 'ls':
         1.309.712   instructions # 0,71 insnt per cycle
         1.848.886   cycles
       0,000800891 seconds time elapsed
```

It is important to note that superscalar architectures usually have a limited number of hardware counters. Since each hardware counter can be used to monitor a single event at any given time, the number of events that can be monitored simultaneously is limited.

The perf tool performs a monitoring time-multiplexing technique to monitor a more significant number of events, allowing each event to have a portion of the counter usage time.

However, what this approach makes possible is just an estimate of the actual behavior of the application since the multiplexed events are not monitored all the time exclusively. Thus, each event should be monitored individually through the `perf stat` instruction, performing the application's profiling several times.

4.5.2 NEC FTRACE

Developed by NEC, the FTRACE tool can be used to obtain information from hardware counters present in NEC's vector architecture. It is necessary to recompile the application using the compiler developed by the company NEC to use the tool. For applications written in C language we use `ncc`, for C++ applications we use `nc++` and for Fortran applications we use `nfort`. In addition, you need to add the *-ftrace* compile flag, as shown in the example: `ncc -ftrace source.c`. We can run the application as we would with a regular executable. At the end of the execution, the information file `ftrace.out` is generated, which can be read using the same tool (and directing the output to the file output):

```
ftrace -f ftrace.out >> output
```

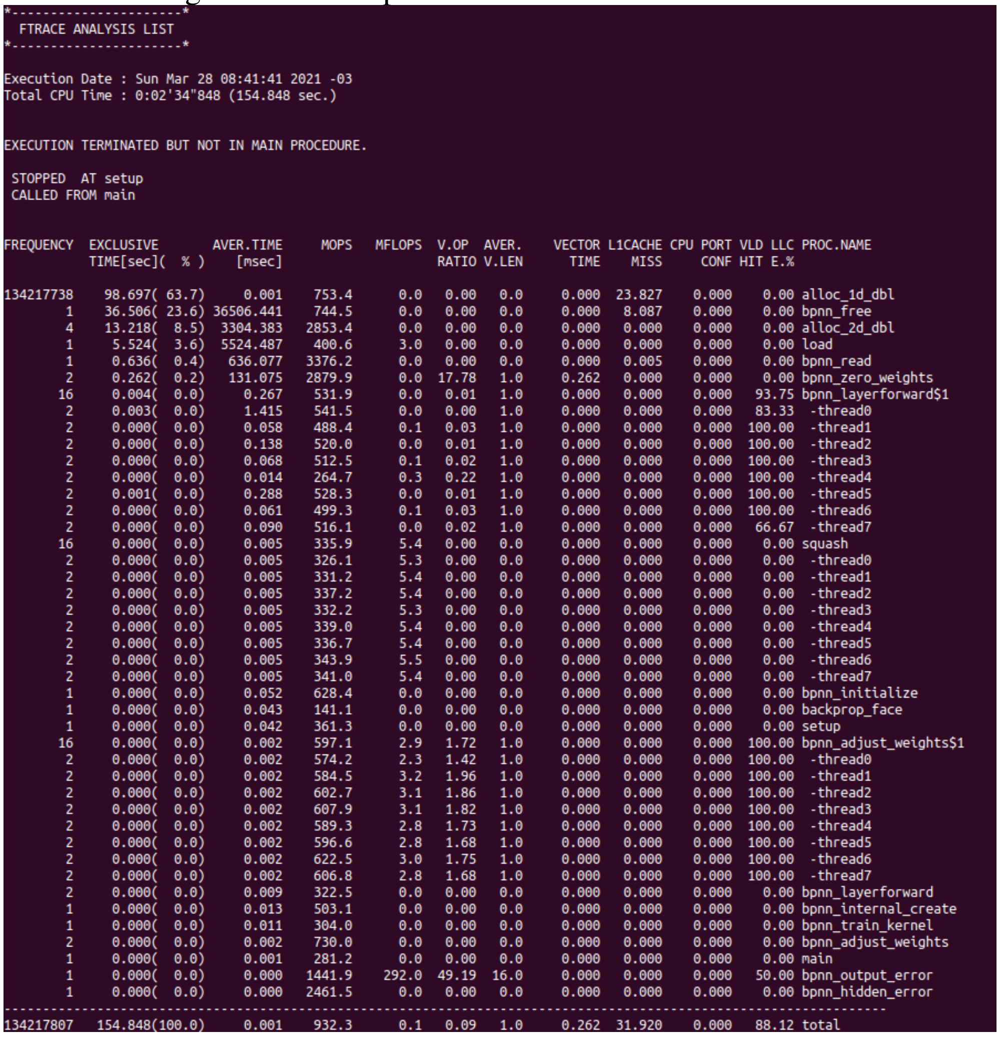
The generated file has a lot of information. Among them, we can highlight the total execution time, the name of the functions executed, the number of times each function was called, the percentage of core utilization by each function, and information about the number of misses of the different levels of cache, among others.

In Figure 4.6 we have a better representation of the output provided by the FTRACE tool. Notably, we can view cache information, total and partial execution time per thread and vectorization information such as vectorization time, vectorization rate and the average size of the created vectors.

It is also possible to control the type of information desired through the two different profiling modes indicated through the `VE_PERF_MODE` environment variable. If the variable has the value `VECTOR-OP` or is undefined (assuming the default value), FTRACE generates information mainly related to vector instructions. If the value of the variable `VE_PERF_MODE` is `VECTOR-MEM`, the data collected correspond mainly to memory accesses. In this way, it can be beneficial to use both modes and aggregate your results at the end. You can change the value of the environment variable `VE_PERF_MODE`, to one of two modes, as follows:

```
export VE_PERF_MODE=VECTOR-OP
export VE_PERF_MODE=VECTOR-MEM
```

Figure 4.6: Example results of the FTRACE command



```
*---------------------*
  FTRACE ANALYSIS LIST
*---------------------*

Execution Date : Sun Mar 28 08:41:41 2021 -03
Total CPU Time : 0:02'34"848 (154.848 sec.)


EXECUTION TERMINATED BUT NOT IN MAIN PROCEDURE.

 STOPPED  AT setup
 CALLED FROM main


FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                    RATIO V.LEN      TIME    MISS     CONF HIT E.%

134217738    98.697( 63.7)     0.001    753.4      0.0   0.00   0.0     0.000  23.827    0.000    0.00 alloc_1d_dbl
        1    36.506( 23.6) 36506.441    744.5      0.0   0.00   0.0     0.000   8.087    0.000    0.00 bpnn_free
        4    13.218(  8.5)  3304.383   2853.4      0.0   0.00   0.0     0.000   0.000    0.000    0.00 alloc_2d_dbl
        1     5.524(  3.6)  5524.487    400.6      3.0   0.00   0.0     0.000   0.000    0.000    0.00 load
        1     0.636(  0.4)   636.077   3376.2      0.0   0.00   0.0     0.000   0.005    0.000    0.00 bpnn_read
        2     0.262(  0.2)   131.075   2879.9      0.0  17.78   1.0     0.262   0.000    0.000    0.00 bpnn_zero_weights
       16     0.004(  0.0)     0.267    531.9      0.0   0.01   1.0     0.000   0.000    0.000   93.75 bpnn_layerforward$1
        2     0.003(  0.0)     1.415    541.5      0.0   0.00   1.0     0.000   0.000    0.000   83.33  -thread0
        2     0.000(  0.0)     0.058    488.4      0.1   0.03   1.0     0.000   0.000    0.000  100.00  -thread1
        2     0.000(  0.0)     0.138    520.0      0.0   0.01   1.0     0.000   0.000    0.000  100.00  -thread2
        2     0.000(  0.0)     0.068    512.5      0.1   0.02   1.0     0.000   0.000    0.000  100.00  -thread3
        2     0.000(  0.0)     0.014    264.7      0.3   0.22   1.0     0.000   0.000    0.000  100.00  -thread4
        2     0.001(  0.0)     0.288    528.3      0.0   0.01   1.0     0.000   0.000    0.000  100.00  -thread5
        2     0.000(  0.0)     0.061    499.3      0.1   0.03   1.0     0.000   0.000    0.000  100.00  -thread6
        2     0.000(  0.0)     0.090    516.1      0.0   0.02   1.0     0.000   0.000    0.000   66.67  -thread7
       16     0.000(  0.0)     0.005    335.9      5.4   0.00   0.0     0.000   0.000    0.000    0.00 squash
        2     0.000(  0.0)     0.005    326.1      5.3   0.00   0.0     0.000   0.000    0.000    0.00  -thread0
        2     0.000(  0.0)     0.005    331.2      5.4   0.00   0.0     0.000   0.000    0.000    0.00  -thread1
        2     0.000(  0.0)     0.005    337.2      5.4   0.00   0.0     0.000   0.000    0.000    0.00  -thread2
        2     0.000(  0.0)     0.005    332.2      5.3   0.00   0.0     0.000   0.000    0.000    0.00  -thread3
        2     0.000(  0.0)     0.005    339.0      5.4   0.00   0.0     0.000   0.000    0.000    0.00  -thread4
        2     0.000(  0.0)     0.005    336.7      5.4   0.00   0.0     0.000   0.000    0.000    0.00  -thread5
        2     0.000(  0.0)     0.005    343.9      5.5   0.00   0.0     0.000   0.000    0.000    0.00  -thread6
        2     0.000(  0.0)     0.005    341.0      5.4   0.00   0.0     0.000   0.000    0.000    0.00  -thread7
        1     0.000(  0.0)     0.052    628.4      0.0   0.00   0.0     0.000   0.000    0.000    0.00 bpnn_initialize
        1     0.000(  0.0)     0.043    141.1      0.0   0.00   0.0     0.000   0.000    0.000    0.00 backprop_face
        1     0.000(  0.0)     0.042    361.3      0.0   0.00   0.0     0.000   0.000    0.000    0.00 setup
       16     0.000(  0.0)     0.002    597.1      2.9   1.72   1.0     0.000   0.000    0.000  100.00 bpnn_adjust_weights$1
        2     0.000(  0.0)     0.002    574.2      2.3   1.42   1.0     0.000   0.000    0.000  100.00  -thread0
        2     0.000(  0.0)     0.002    584.5      3.2   1.96   1.0     0.000   0.000    0.000  100.00  -thread1
        2     0.000(  0.0)     0.002    602.7      3.1   1.86   1.0     0.000   0.000    0.000  100.00  -thread2
        2     0.000(  0.0)     0.002    607.9      3.1   1.82   1.0     0.000   0.000    0.000  100.00  -thread3
        2     0.000(  0.0)     0.002    589.3      2.8   1.73   1.0     0.000   0.000    0.000  100.00  -thread4
        2     0.000(  0.0)     0.002    596.6      2.8   1.68   1.0     0.000   0.000    0.000  100.00  -thread5
        2     0.000(  0.0)     0.002    622.5      3.0   1.75   1.0     0.000   0.000    0.000  100.00  -thread6
        2     0.000(  0.0)     0.002    606.8      2.8   1.68   1.0     0.000   0.000    0.000  100.00  -thread7
        2     0.000(  0.0)     0.009    322.5      0.0   0.00   0.0     0.000   0.000    0.000    0.00 bpnn_layerforward
        1     0.000(  0.0)     0.013    503.1      0.0   0.00   0.0     0.000   0.000    0.000    0.00 bpnn_internal_create
        1     0.000(  0.0)     0.011    304.0      0.0   0.00   0.0     0.000   0.000    0.000    0.00 bpnn_train_kernel
        2     0.000(  0.0)     0.002    730.0      0.0   0.00   0.0     0.000   0.000    0.000    0.00 bpnn_adjust_weights
        1     0.000(  0.0)     0.001    281.2      0.0   0.00   0.0     0.000   0.000    0.000    0.00 main
        1     0.000(  0.0)     0.000   1441.9    292.0  49.19  16.0     0.000   0.000    0.000   50.00 bpnn_output_error
        1     0.000(  0.0)     0.000   2461.5      0.0   0.00   0.0     0.000   0.000    0.000    0.00 bpnn_hidden_error
------------------------------------------------------------------------------------------------------------
134217807   154.848(100.0)     0.001    932.3      0.1   0.09   1.0     0.262  31.920    0.000   88.12 total
```

4.5.3 Score-P

The instrumentation of the MARE2DEM application uses Score-P. Our first step was to identify which sections of the source code implement the forward response computation, which contains the Finite Element Modeling (FEM), by the worker processes, considering a refinement group. Since we knew that the forward response parallel computation is implemented using the MPI interface, we started by searching for MPI calls inside MARE2DEM's source code, mainly implemented in FORTRAN.

Inside the source code, a subroutine is implemented, which contains the worker's logic of which task to perform, according to the message received by the manager process. Inside is the call path that leads to the forward response computation. With this in mind, it was utilized the manual region instrumentation (KNÜPFER et al., 2012) functionality of Score-P to track the timestamps of the call sequence of all workers. This sequence was grouped into a single manual Score-P region and labeled as compute. Compute is the main MARE2DEM computing phase that will be analyzed since it encapsulates the majority of the parallel MPI computation performed by the workers.

5 INVESTIGATION AND RESULTS

The results section will present the main findings obtained by this dissertation. The following section is regarding the results of the first part and the optimization techniques implemented in the NAS benchmark and RTM. The second section refers to the second part, which is the performance results of MARE2DEM's adaptation for the SX-Aurora. Finally, the outcome of the third part of this dissertation is the adaptation of Hiperwalk for NEC's vector architecture.

5.1 Optimization with loop unrolling and inlining

The experiments and their results are presented below. Initially, the experiments and analysis results are exposed, with the optimizations implemented. Finally, a discussion of the results obtained, comparing the optimized and the original. Repeating for both the benchmark NAS and the RTM modeling.

5.1.1 NAS Benchmark inlining and loop unrolling

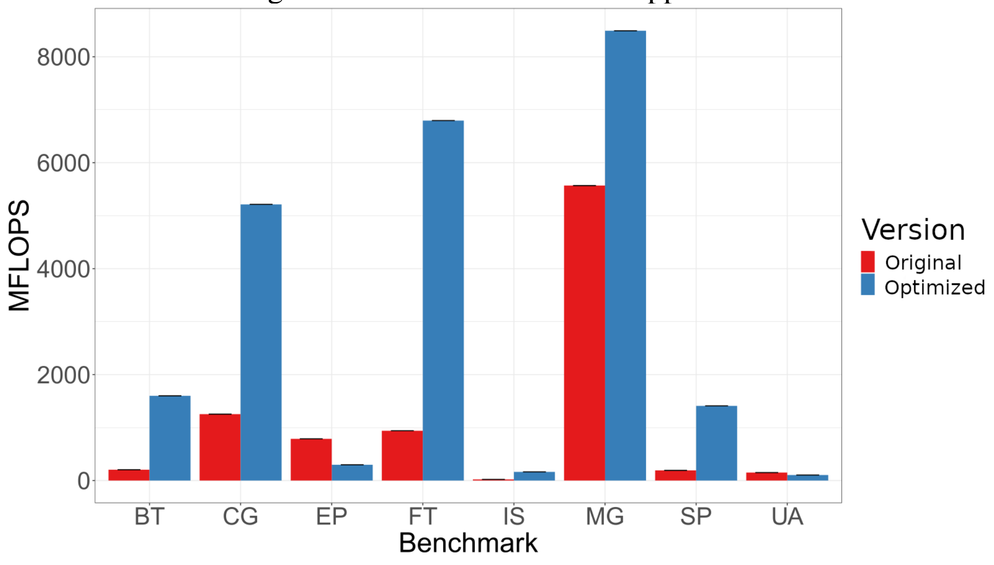
Thus, applying the previously mentioned optimization techniques, inlining and loop unrolling, in the benchmark NAS, and comparing it with the original implementation, we obtain Figure 5.1, being the x axis each application of the benchmark NAS used in this test, the y axis shows the FLOPS, the red bars corresponds to the original implementation and the blue bars represents the optimized version.

It is noticed that the FLOPS increase considerably, reaching up to $7.8\times$ higher for the optimized application of the benchmark BT. Comparing the increase between the original and the optimized version, we have an average of 204.42 MFLOPS and 1599.18 MFLOPS, respectively.

Also, it is necessary to point out that in two tests the optimization techniques had a negative impact. Those were the EP and UA. The negative impact regard EP is due to it being, as the name suggests, Embarrassingly Parallel, which the compiler is more then capable to optimize 100%. However we still implemented manually the optimizations to see if there was a possibility to improve.

However in the UA case, the problem is different. Due to its unstructured nature,

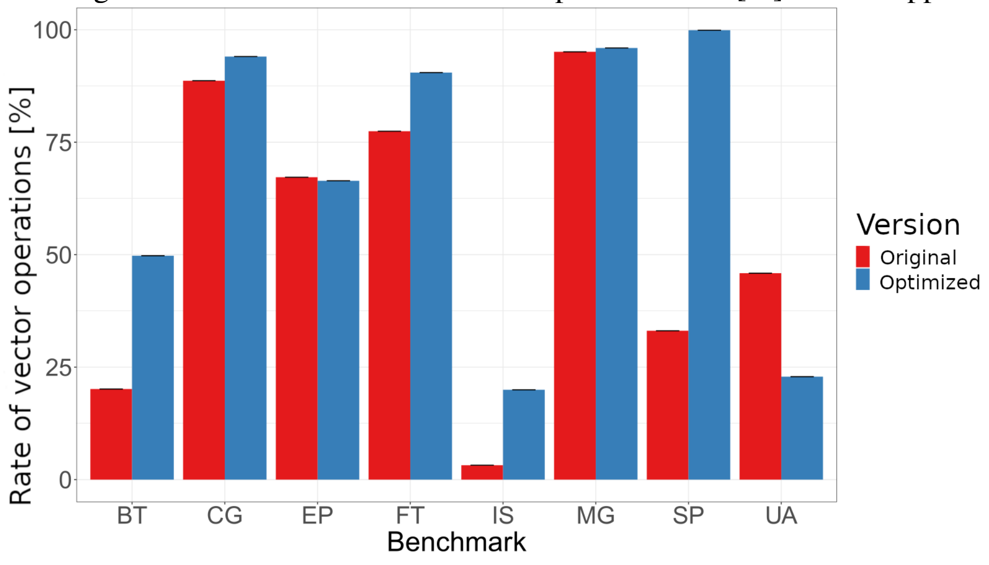
Figure 5.1: MFLOPS X NAS Application



its incredible difficult to implement these kind of optimizations. Still, we tried to implement them, as it was with EP. Therefore, we were able to illustrate a point that we must be careful where we apply optimization prematurely.

In Figure 5.2 we have the rate of vector operations on the y axis for each application of the benchmark NAS on the x axis, with the original applications in red and the optimized version in blue. The BT program in the optimized code has a higher rate of vector operations than the original, approximately from 20% to 50%, more than twice as many vector operations. This large increase in operations is due to loop unrolling and inlining helping the compiler in its automatic vectorization. Furthermore, for the BT program, for example, the average size of each vectorization also increased by approximately 31%, going from 92 elements to 120 elements.

Figure 5.2: NAS benchmark: Vector operation rates [%] X NAS app



Furthermore, it is worth mentioning that the cache memory also benefited from the optimizations, again, taking the BT program as an example, an increase of approximately 10% in the success rate of cache L3 was obtained.

Again, here we can see the UA and EP had their MFLOPs decreased due to the decline in the rate of vectorization.

5.1.2 RTM optimization with inlining and loop unrolling

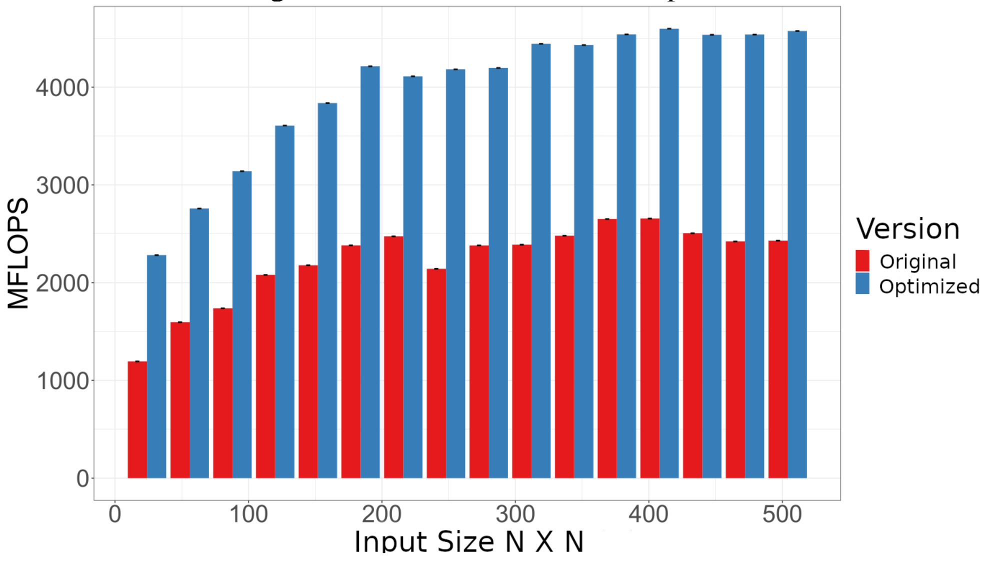
Next, the experiment using RTM modeling is presented, performed with the original version OpenMP. The configuration used for the RTM application is the same as for the optimization with the NAS benchmark. Again, the compilation makes use of NEC's exclusive compiler and the same flags used in the previous experiment. Like the NAS benchmark, the "original" version matches the unmodified RTM code applied to the automatic vectorization of the SX-Aurora TSUBASA machine. The optimized version is equivalent to the implementation of the techniques presented above, inlining and loop unrolling.

Like NAS optimization, data collection uses the NEC profiling program PROGINF. Through this program, various information related to the architecture and execution, mainly the FLOPS and the cache memory failure rate.

Again, the compiler makes a complete report containing various information regarding the compilation. For example, to improve vectorization, loop unrolling was used in several functions. Again, the guide provided by NEC (NEC, 2020a) and the previously mentioned techniques were used to optimize the application.

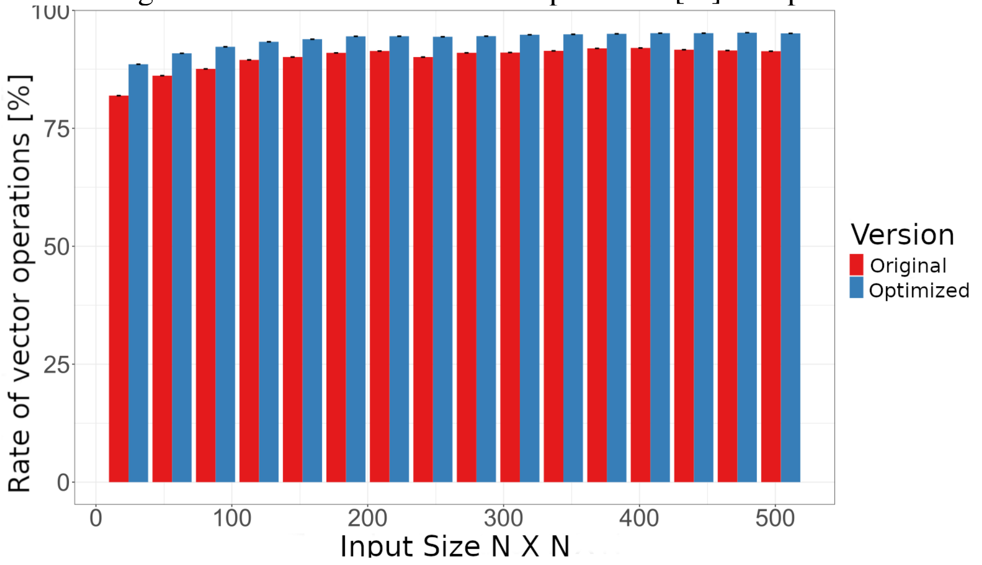
Therefore, Figure 5.3 shows the experiment applied to the RTM application, showing the FLOPS on the y axis concerning the input size on the x axis, with the optimized version in blue bars and the optimized version in red bars. Original version. FLOPS vary considerably from most minor to most significant entry. Looking at the original version and the new optimized version, for the most prominent input 504×504, we have 2429.83 MFLOPS and 4574.75 MFLOPS, respectively, an increase of approximately $1.9\times$. This significant increase is mainly due to the decrease in the total number of instructions by approximately 45%, from $15 \cdot 10^{10}$ instructions to $89 \cdot 10^{9}$ instructions. This is due to the automatic vectorization done by the NEC compiler, enhanced by the applied techniques of loop unrolling and inlining. It should also be noted that loop unrolling has decreased the number of non-vectorized execution instructions.

Figure 5.3: RTM: MFLOPS X Input Size



The memory hit rate cache is shown in Figure 5.4. On the y axis, we have the success rate of the cache L3 memory (LLC) about the input size on the x axis, with the optimized version in blue bars and the original version in red. Comparing the biggest data entry 504×504 for the original version and the new optimized version, we have an increase of only 1.5% or so. Although the comparison shows only a small increase for the success of the cache L3 memory, analyzing the cache L1, we have a 50% decrease in the time of failure of the cache L1 memory, decreasing from 10 seconds to 5 seconds approximately. Due to the loop unrolling technique, the loop instruction overheads is reduced, and inlining minimizes the function call instructions.

Figure 5.4: RTM: Rate of vector operations [%] X Input Size



5.2 NEC MARE2DEM Implementation

The following result to be shown is regarding the first experiment. It uses the artificial demonstrative data set provided along with MARE2DEM's application. The following result utilizes the same data set but different worker and refinement group configurations. The last result comes from the third experiment utilizing the real-world data set provided by Petrobras.

5.2.1 Outcome validation

After the modifications were done to the math libraries exclusive to Intel, a first experiment was performed. We needed first to be sure that all the modifications had no repercussions on the final result regarding the output of MARE2DEM's execution, which is shown by Figure 5.5.

Figure 5.5 shows the visualization of MARE2DEM final iteration. The graph on the top shows the execution of an x86 architecture. On the bottom graph, the SX-Aurora execution is shown. These graphs provide the resistivity regarding the depth of the ocean floor.

Figure 5.6 also shows the MARE2DEM's outcome, now related to the final iteration of the real-world data set. The top graph shows the x86 and the bottom one the SX-Aurora implementation.

Finally we can make a subtraction of both results and show the percentage difference in both results, in Figure 5.7 and Figure 5.8.

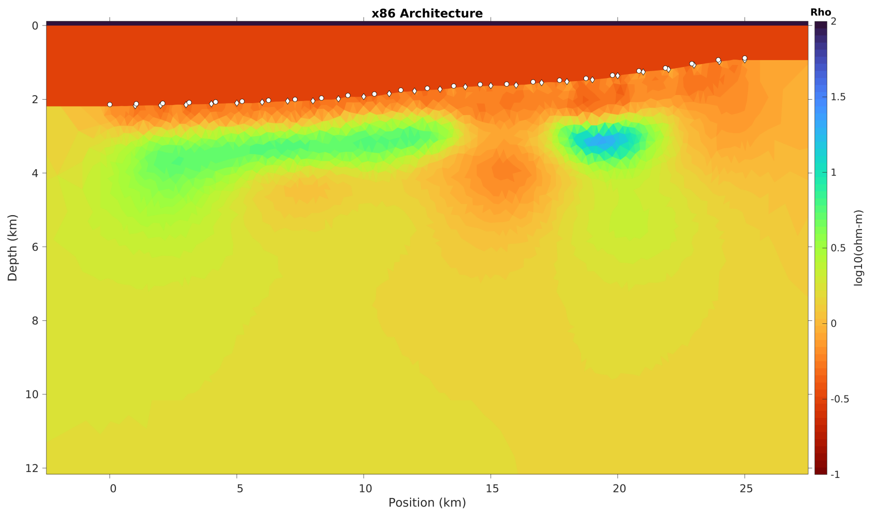
Figure 5.7 gives us the resistance percentage difference in each triangle regarding the demonstration data set. The highest difference is 3,6%.

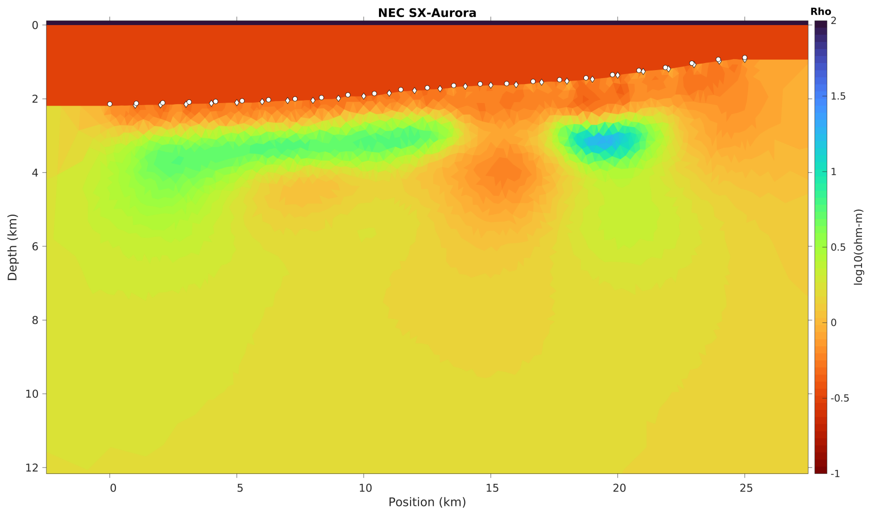
In regards the Petrobras data set, Figure 5.8 gives us the resistance percentage difference in each triangle. Peak percentage difference is 6%.

Besides the small percentage differences and the naked eye comparison, to further validate the claim that the modification made to the MARE2DEM code had no adverse effect, Figures 5.7 and 5.8 must be compared to the resistance difference in an execution of only the unmodified code. So, this test is shown on Figures 5.9 and 5.10.

Similarly as before, Figures 5.9 and 5.10 gives us the resistance percentage difference in each triangle, for the demo and Petrobras datasets, respectively. Visually they are identical to Figures 5.7 and 5.8. However there is a small percentage difference. In Figure

Figure 5.5: Demonstration data set comparison for outcome validation between x86 and SX-Aurora.





5.9, the highest percentage difference is 3,8%. In Figure 5.10, the difference is smaller than before, only 5,9%.

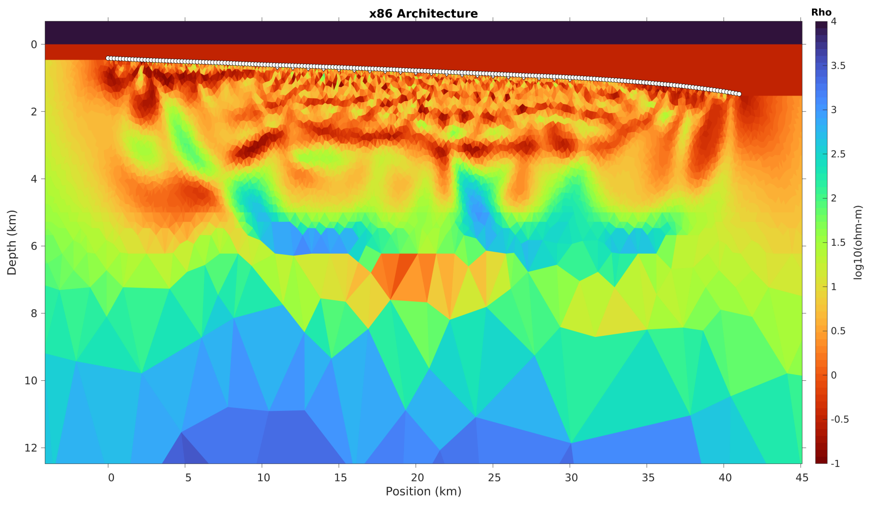
Even though there are differences, this only shows that between executions of the same dataset on the same unmodified code, there is a similar difference between executions. This is due to the nature of iterating with 2D inversion and adaptive refined elements. Therefore we can assume that the modifications implemented, the NEC's math library and all minor changes had no impact on the final result.

5.2.2 Initial artificial demonstration runtime

After assessing the impact on the final results, we can move forward with the execution's investigation. Next, Figure 5.11 shows the execution of the artificial dataset on the x86 architecture.

The x86 architecture execution represented in Figure 5.11 shows the runtime, in seconds, for every 23 workers. We can see that the final runtime was 206 seconds. The image shows the expected behavior of one iteration of MARE2DEM's execution. There is

Figure 5.6: Comparison for outcome validation between x86 and SX-Aurora, related to the real world data set.



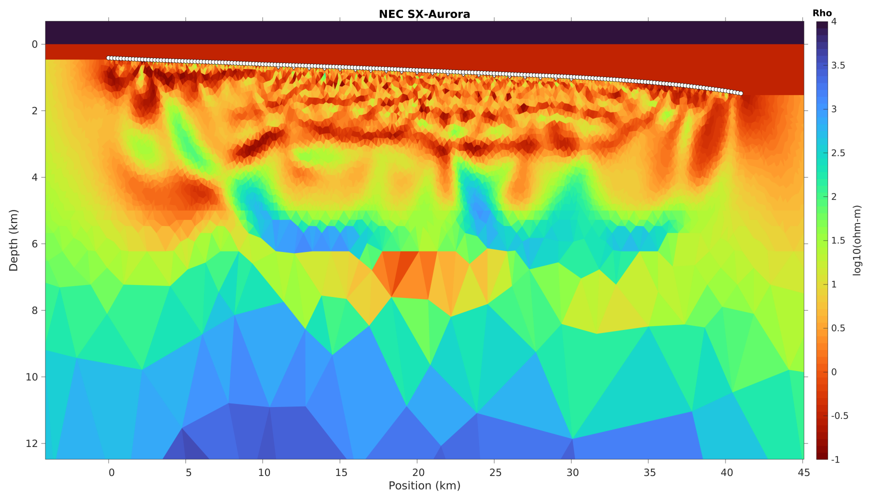


Figure 5.7: Resistance difference in percentage, for the Demonstration data set outcome between SX-Aurora X x86.

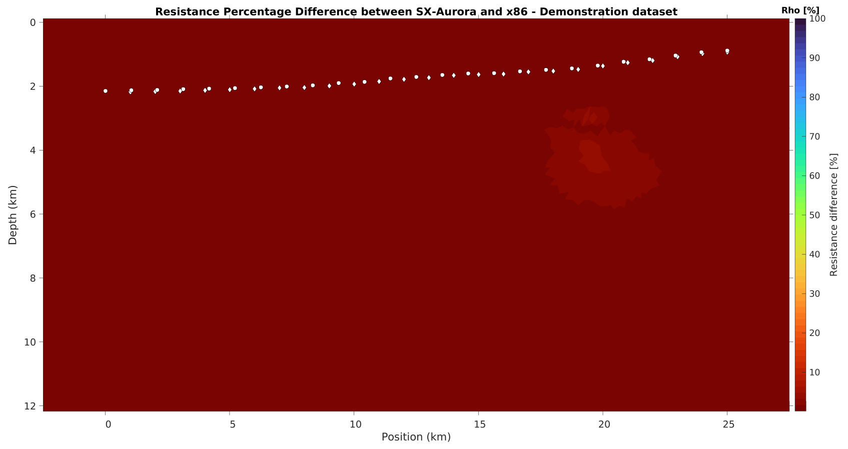


Figure 5.8: Resistance difference in percentage, for the Petrobras data set outcome between SX-Aurora X x86.

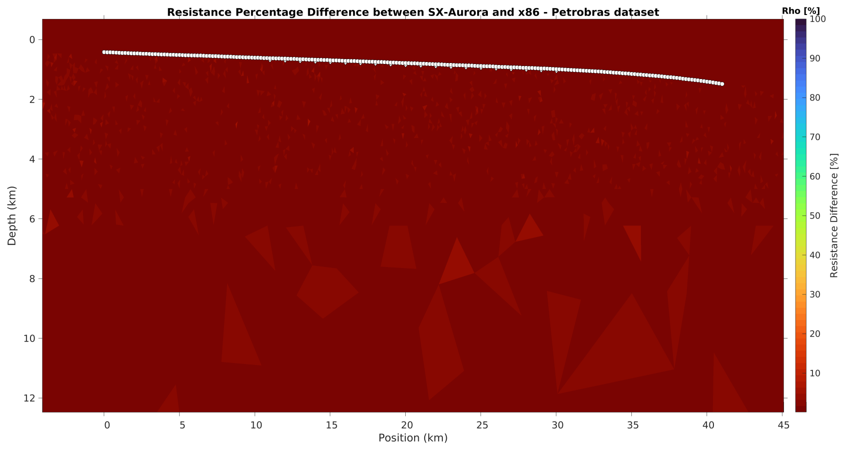
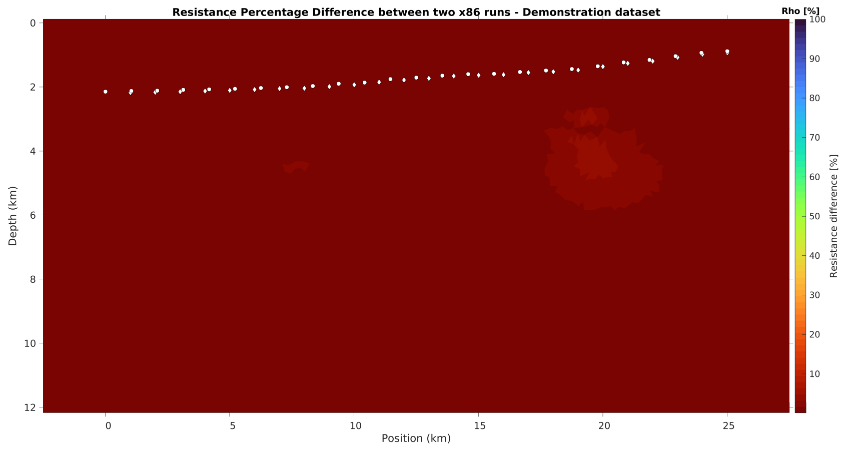


Figure 5.9: Resistance difference in percentage, for the Demonstration data set outcome between two x86 executions.



a clear division between left and right on these runtime graphs. They denote two different stages of one iteration, as the manager needs all workers to end their load to start the

Figure 5.10: Resistance difference in percentage, for the Petrobras data set outcome between two x86 executions.

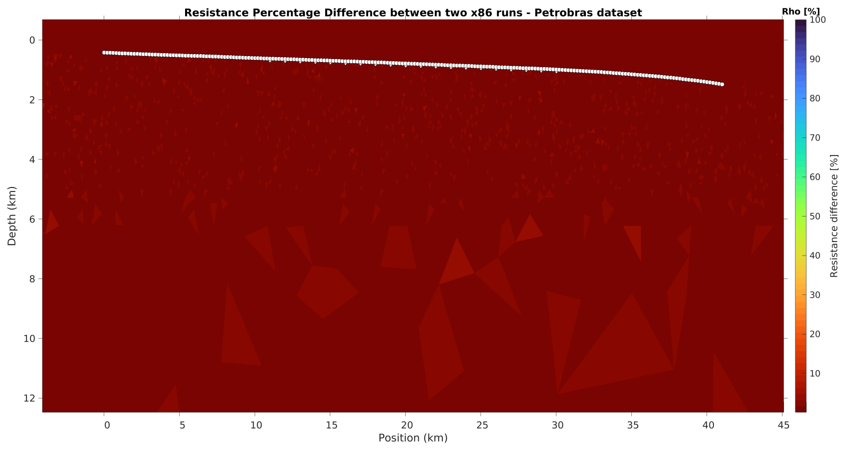
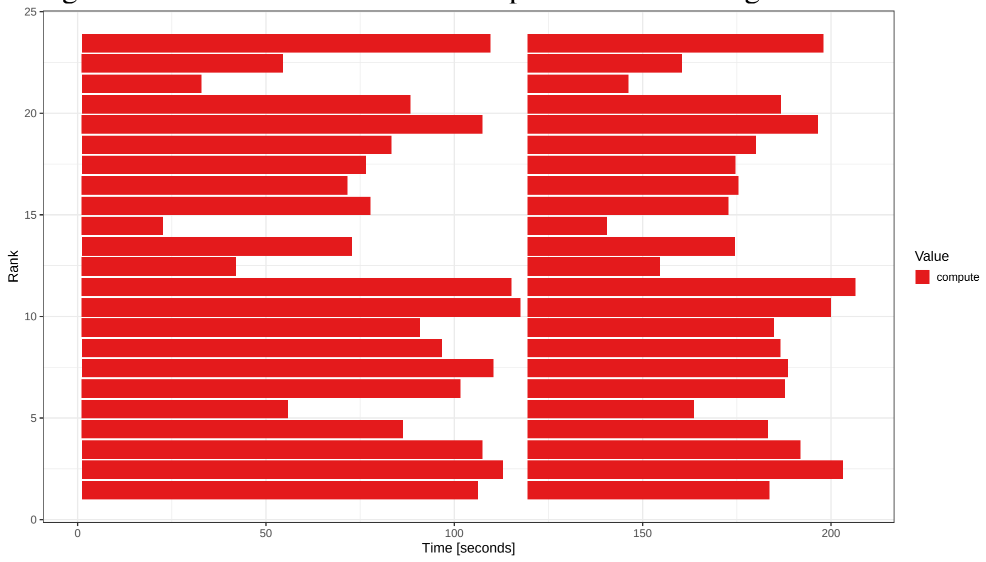


Figure 5.11: Results in seconds of experiment 1 utilizing x86 architecture



second stage.

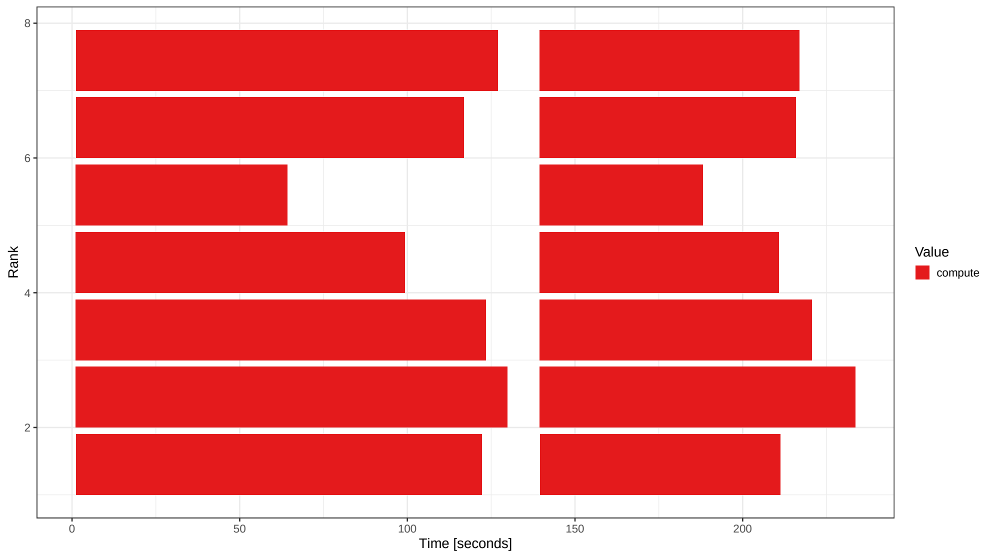
Following the next result, we have Figure 5.12. Here we execute the first data set, the artificial one, on NEC's SX-Aurora.

Identical to the previous graph, Figure 5.12 gives the runtime for each of the seven workers. The final runtime was 237 seconds. So, the SX-Aurora performed slower than the x86 architecture, around 15% slower.

Both Figure 5.12 and Figure 5.11 show the traces collected via Score-P, which can be hard to understand at a glance. To facilitate the comparison between both runtimes, Figure 5.13 is presented.

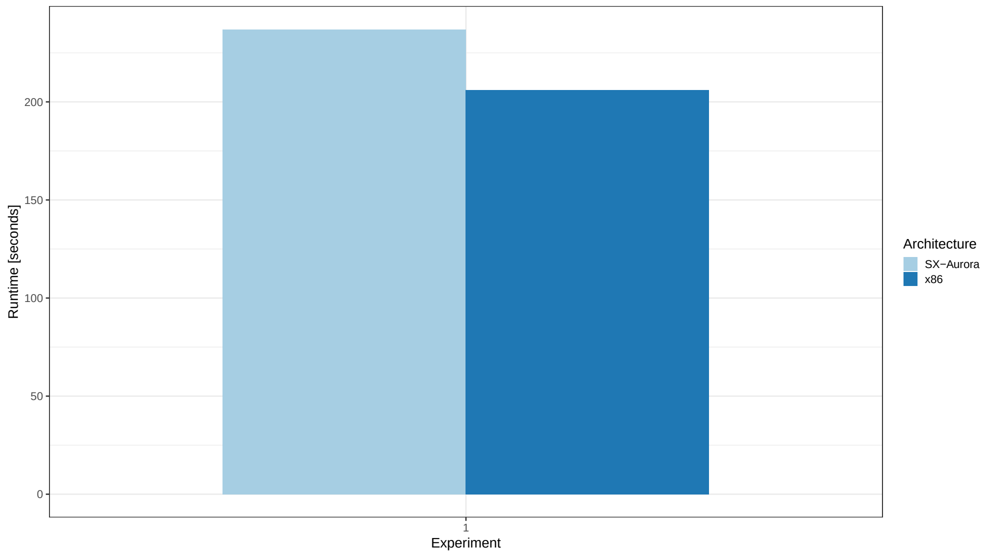
Figure 5.13 shows on the x-axis the experiment being run, while the y axis represents the runtime of the application in seconds. The light blue bar refers to the SX-Aurora architecture and the darker blue to the x86 architecture. It is easier now to see the differ-

Figure 5.12: Results in seconds of experiment 1 utilizing the SX-Aurora



ence in runtimes of both platforms.

Figure 5.13: Experiment 1 comparison between x86 and SX-Aurora runtimes



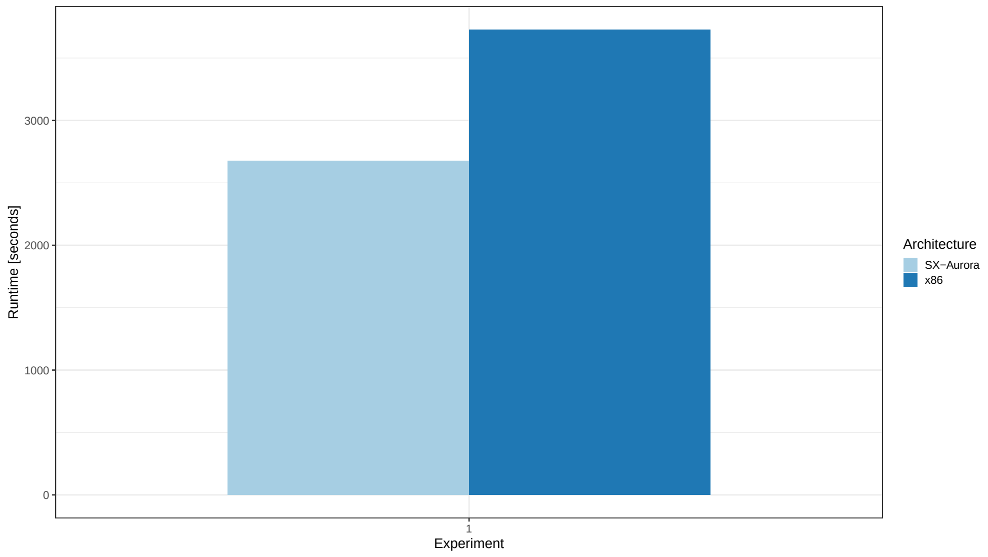
Initially, it was expected for the SX-Aurora to improve MARE2DEM's performance already. However, upon further investigation, especially if the data set was proper, it was attested that the amount of data being transferred was small for each worker, which goes against the advertised advantages of this architecture. SX-Aurora has a very high memory transfer rate, and this power was not being harvested in this data set, resulting in a negative result.

5.2.3 Effects of one worker on the first data set

After the enlightenment of the last result's investigation, a second experiment was proposed, utilizing the same dataset. A new worker and refinement group configuration was chosen in this second one, so it is possible to appropriate the advantages of the SX-Aurora architecture.

Hence, the configuration to best take advantage of SX-Aurora high memory transfer was to use only one worker, besides the manager, and one refinement group. Theoretically, it would allow for a larger-sized data set and higher data transfers during the MARE2DEM execution. Figure 5.14 shows these results.

Figure 5.14: Experiment 2 comparison of the runtimes between x86 and SX-Aurora.



In Figure 5.14 we have a comparison of MARE2DEM's execution on the x86 and SX-Aurora architectures. The light blue bar refers to the SX-Aurora execution, and the dark blue line refers to the x86. Again, similar to the previous two graphs, on the x-axis is displayed the execution runtime. On the y axis are the two different architectures.

It is easier to compare both architectures' runtimes now. We can see that SX-Aurora can improve the runtime without supporting multiple cores and more significant data transfers. The x86 architecture execution runtime was of 3728 seconds approximately. SX-Aurora runtime reached around 2677 seconds. An increase in performance of approximately 39%.

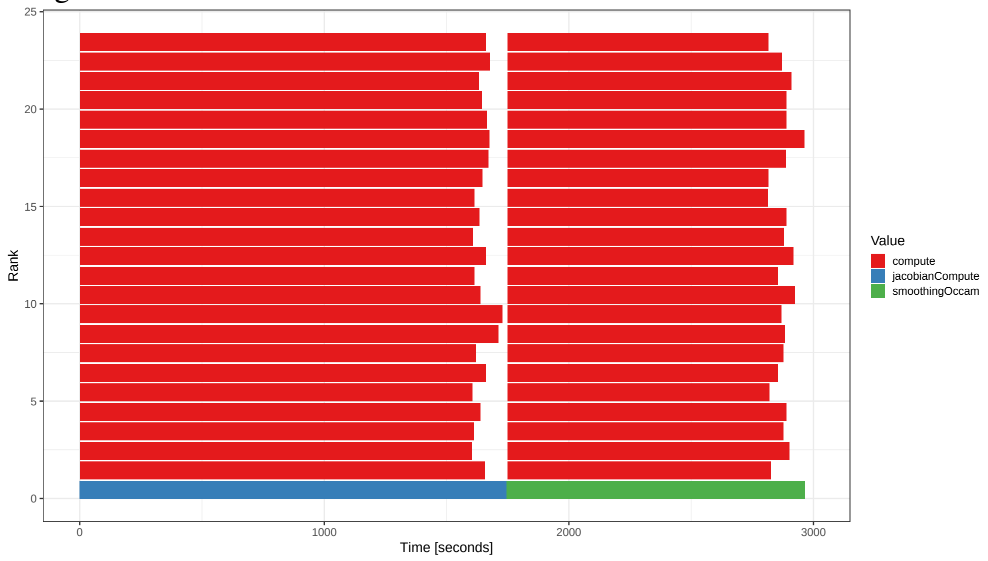
This experiment shows that the SX-Aurora architecture has potential for performance gains. However, the experiment does not reflect any real-world use case, as both platforms are underutilized, which is evident in the total runtime of both experiments, boasting an increase in execution runtime of around 17 times more significant in the worst case.

5.2.4 Real world data set

As the last experiment was a test to investigate the potential of SX-Aurora architecture, the last experiment will put this potential to test in a real-world scenario. This data set, provided by Petrobras, is larger than the previous one by roughly 50 times the amount of data.

Figure 5.15 shows a similar graph as before. We have on the x-axis the runtime in seconds. The y axis shows the workers. Particularly on this graph, we have the presence of the manager, illustrated by the first worker. This graph regards the execution of the real-world data set on the x86 architecture, achieving, for one iteration, an execution runtime of 2964 seconds approximately.

Figure 5.15: Runtime for real world dataset - 1 Iteration - x86 architecture.



Next, Figure 5.16 has the execution of the MARE2DEM application on the SX-Aurora architecture. In the last graph, the x-axis and y-axis are the runtimes in seconds and the workers. The final runtime is around 2341 seconds for one iteration.

To better represent the comparison between runtimes, Figure 5.17 is shown. The x-axis represents the experiment, while the y-axis shows the application runtime in seconds. The light blue refers to the SX-Aurora architecture and the darker blue to the x86 architecture. Here it is possible to visualize the last statistic, and the SX-Aurora impacts the performance by around 27%.

Figure 5.16: Runtime for real world dataset - 1 Iteration - SX-Aurora.

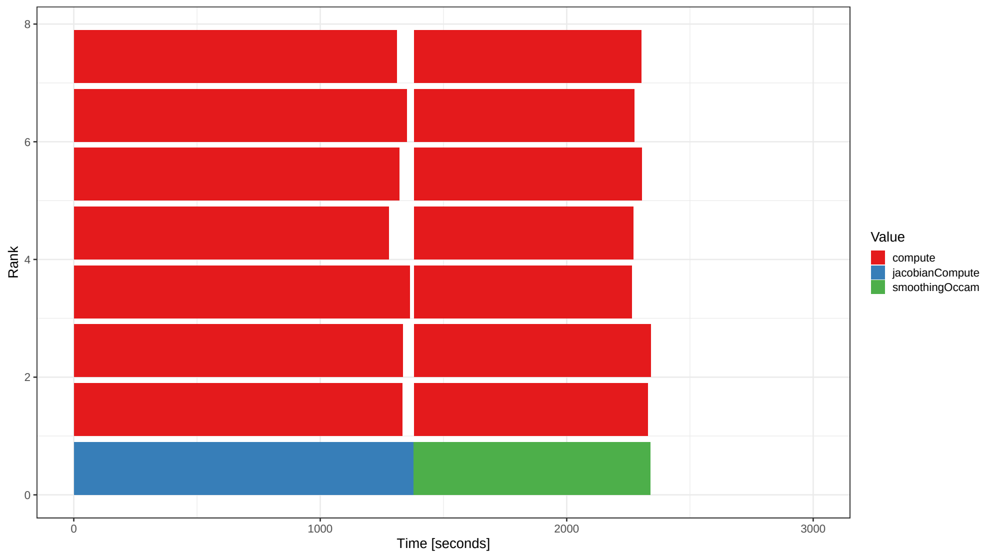
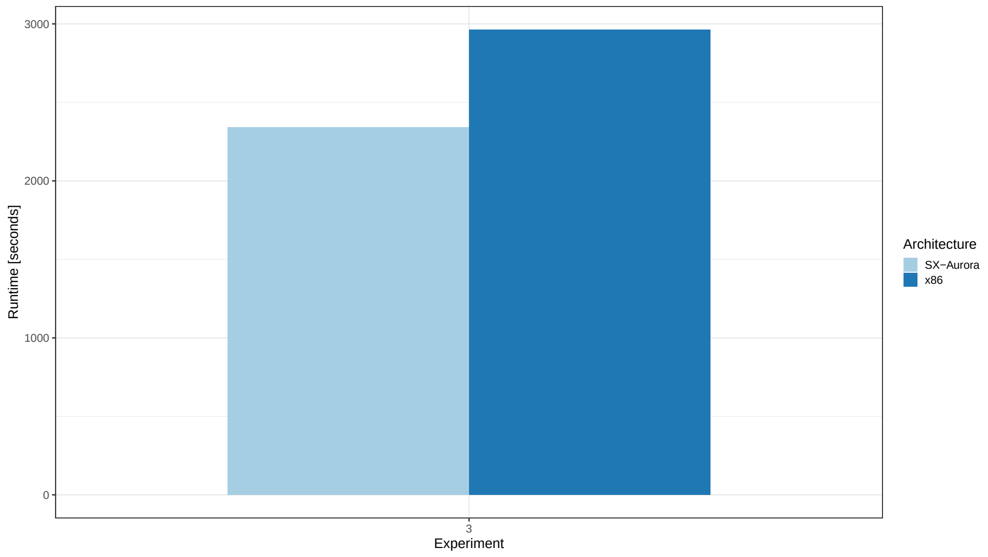


Figure 5.17: Comparison of experiment 3 for x86 and SX-Aurora runtimes

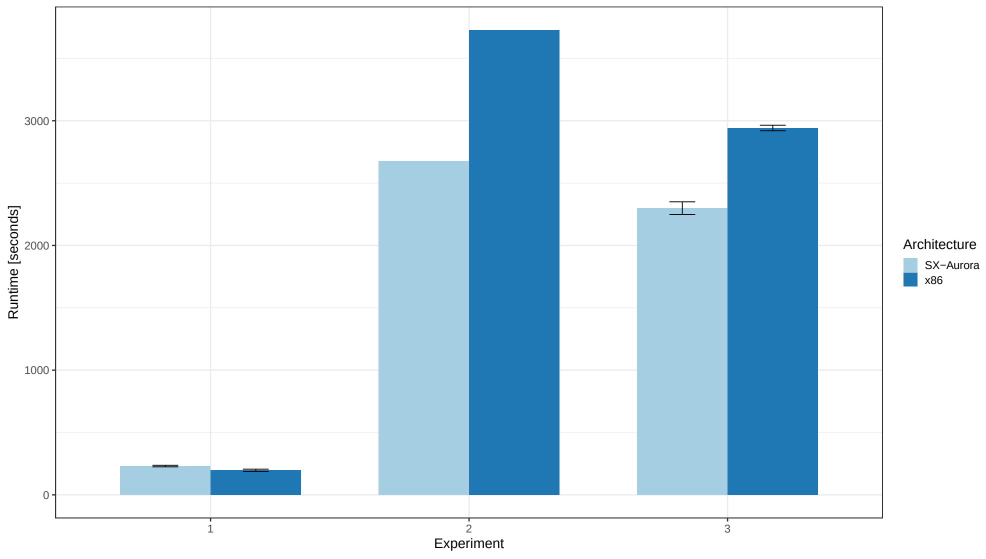


Now, the last illustration, Figure 5.18, elucidates and realizes some of the potentials that NEC's SX-Aurora has under the right circumstances, especially when we can take advantage of its inherent positive traits, like its vector architecture, which can perform multiple of the same calculation with only one instruction and, in this case specifically, the high memory bandwidth available.

Figure 5.18 shows in the y axis the runtime of each experiment. The x-axis displays the number of all experiments that achieved the runtimes exhibited on the y axis. The light blue color gives the SX-Aurora architecture, and the darker blue represents the x86 architecture.

We employed ten executions in the first and last experiment to improve the statistical rigor to have the mean execution time and error. It is visible that the mean execution

Figure 5.18: Runtime comparison between x86 and SX-Aurora for all three experiments.



time is similar to the three experiments performed, approximately maintaining the differences in runtime between platforms.

This graph, Figure 5.18 illustrates the path taken by this work. We began the runtime evaluation with worse performance for SX-Aurora. However, after analyzing and evaluating the advantages of the architecture, the two subsequent experiments utilized and realized the potential that SX-Aurora has when harvesting its potential.

5.3 Hiperwalk's improvements with vectorization

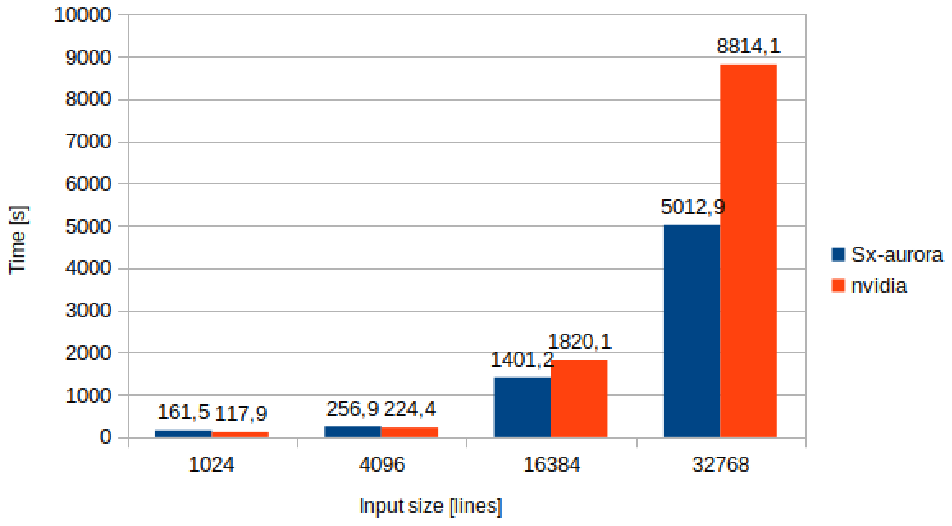
This section will show the improvement vector architectures can make in quantum walk simulators. The first result is the average runtime from 20 executions of each matrix, namely 1024, 4096, 16384, and 32768 nonzero elements. It shows both architectures, NVIDIA and NEC Hiperwalk's implementation.

It is prudent to note that significant code differences in the Neblina math library will necessarily impact performance, as running the HiperWalk simulator in SX-Aurora was the main point.

These results were obtained in seconds. The runtimes of the two NEC and NVIDIA architectures for the four input sizes, 1024, 4096, 16384, and 32768, respectively, are 161,85s, 256,99s, 1401,21s and 5021,94s in regards to SX-Aurora, and NVIDIA's performance is 117,9s, 224,4s, 1820,1s, and 8814,1s. Figure 5.19, which shows these results, will be a recurring reference onward due to its significance.

It is noticed in Figure 5.19 that by increasing the number of nonzero input enti-

Figure 5.19: Runtime for each matrix input size: 1024, 4096 and 16384 and 32768 nonzero entities



ties, NEC's architecture begins to gain with NVIDIA's architecture. The 16384 use case achieved an average of roughly 24% decrease in runtime, from 1820,1 seconds on the P100 to 1401,2 seconds on the SX-Aurora.

The adaptation decreased around 43%, a performance gain of around 75%, achieving approximately 8814,1 and 5012,9 seconds, for the P100 and SX-Aurora, respectively, in the most extensive data set.

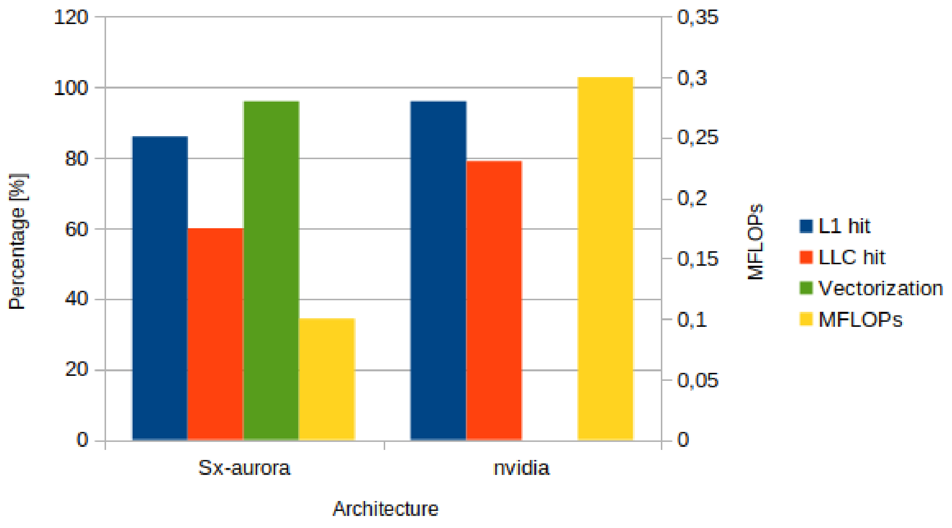
These results meet our expectations. NEC's architecture could not utilize its main strength with low nonzero elements. It is evident that only when a large amount o data is transferred SX-Aurora surpass NVIDIA's architecture.

Figure 5.19 is the main result our objectives were expected to accomplish. To better elucidate the reasons for these improvements, other metrics were extracted.

As mentioned in the methodology, Figure 5.20 shows a graph where the x-axis provides the architecture in question. The y axis shows the relevant unit; in the left, we get the percentage[%], and the right y-axis shows Floating Point Operations per second.

Going back to Figure 5.19 the first data set, 1024 nonzero elements, has significantly poorer performance than the original implementation. Evidence shown in Figure 5.20, points toward what is expected. With small amounts of data provided by the matrix, NEC's SX-Aurora cannot surpass the original OpenCL implementation. SX-Aurora vectorized a great deal of the code, around 96%. Its vector length was extensive, with 131 units. Its LLC cache hit rate was 60%, contrasting with NVIDIA's 80%. However, the L1 cache hit rates were similar, with NEC's 85% and NVIDIA's 96%. In terms of raw performance, the SX-Aurora architecture achieved 0,11 MFLOPs and the P100 architecture

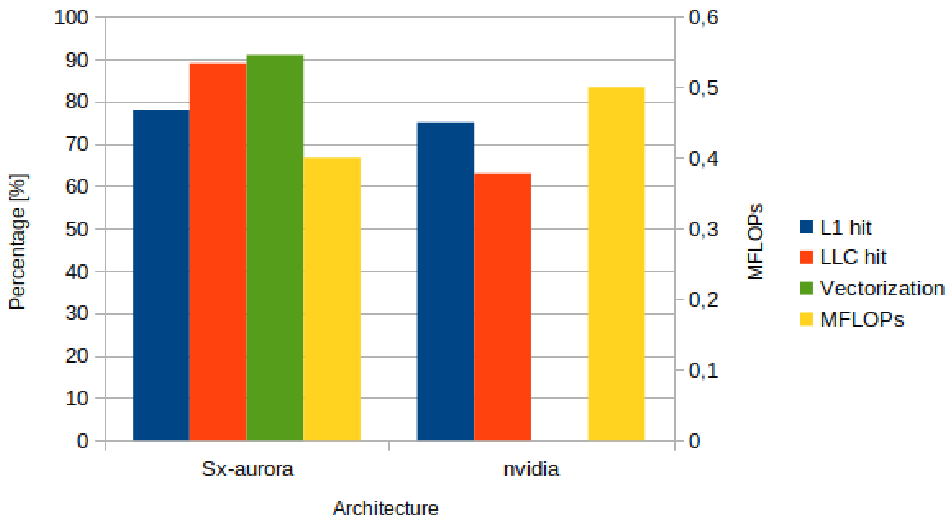
Figure 5.20: Vectorization [%], LLC and L1 cache hit ratio [%] and MFLOPs for the smaller input size of 1024 nonzero elements



around 0,29 MFLOPS.

The second data set results are shown in Figure 5.21. Here we have the same x-axis that provides the particular architecture and the y-axis the relevant unit, the percentage[%] to the left and the right y-axis the Floating Point Operations per second.

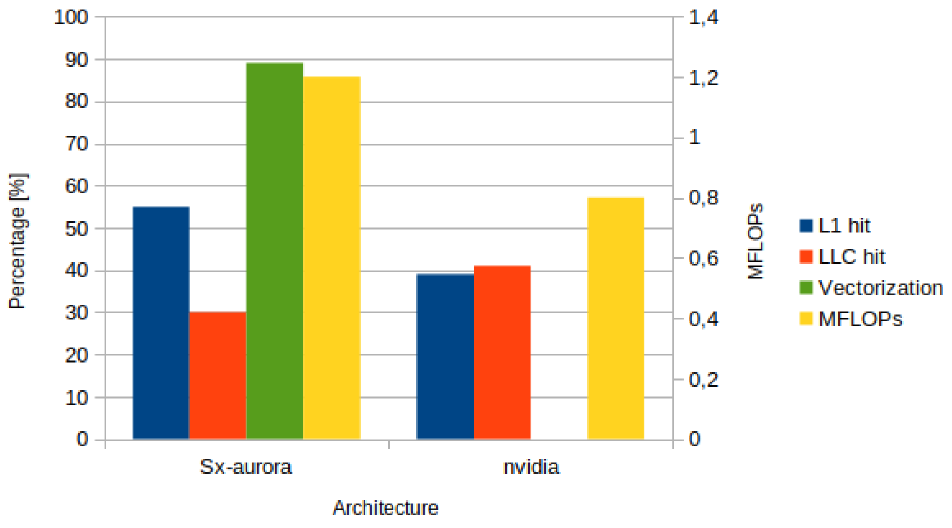
Figure 5.21: Vectorization [%], LLC and L1 cache hit ratio [%] and MFLOPs for the smaller input size of 4096 nonzero entities



Taking another look at Figure 5.19, the second data set with 4096 nonzero entities has a very similar performance to the original implementation. As expected, 5.21 shows again that we still have not enough data to utilize SX-Aurora's potential fully. SX-Aurora was able to vectorize the code, around 91%. The vector length reached 124 units. Its LLC cache hit rate was 90%, contrasting with NVIDIA's 63%. However, the L1 cache hit rates were similar, with NEC's 78% and NVIDIA's 75%. In terms of raw performance, the SX-Aurora architecture achieved 0,39 MFLOPs and the P100 architecture around 0,51 MFLOPS.

The third data set results are presented in Figure 5.22. The x-axis provides the architecture in question. The left y-axis is percentage[%], and the right y-axis shows Floating Point Operations per second.

Figure 5.22: Vectorization [%], LLC and L1 cache hit ratio [%] and MFLOPs for the smaller input size of 16884 nonzero elements

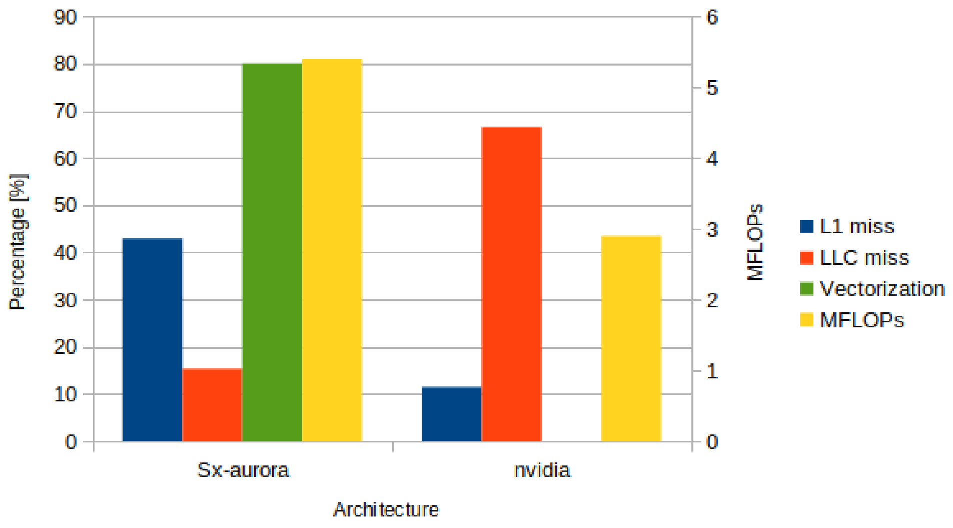


The third data set, 16884 nonzero elements, shows substantial performance increases over the original implementation. Figure 5.22 show pieces of evidence pointing toward the advantages of SX-Aurora. With more significant data NEC's SX-Aurora can surpass the original OpenCL implementation. SX-Aurora vectorized around 89% of the code. Its vector length was extensive, with 176 units. However, the LLC cache hit rate was much lower, around 30%, contrasting with NVIDIA's 41%. L1 cache is different, with NEC's 55% and NVIDIA's 39%. In terms of raw performance, the SX-Aurora architecture achieved 1,23 MFLOPs and the P100 architecture around 0,81 MFLOPS.

The fourth data set results are presented in Figure 5.23. The x-axis provides the architecture in question. The left y-axis is percentage[%], and the right y-axis shows Floating Point Operations per second.

Now, the last illustration, Figure 5.23, elucidates and realizes some of the potentials that NEC's SX-Aurora has under the right circumstances, especially when we can take advantage of its inherent positive traits, like its vector architecture, which can perform multiple of the same calculation with only one instruction and, in this case specifically, the high memory bandwidth available.

Figure 5.23: Vectorization [%], LLC and L1 cache hit ratio [%] and MFLOPs for the smaller input size of 32768 nonzero entities



In Figure 5.19, the fourth and last data set, with 32768 nonzero entities, has improved performance even further. Again, looking at Figure 5.23, provides ample evidence for this runtime improvement. Vectorization in SX-Aurora was able to reach around 80%. Its vector length was close to SX-Aurora's limits, with 223 units of 256. Comparing cache hit ratios, SX-Aurora had only 15% LLC cache hit rate and 43% for the L1 cache hit ratio. NVIDIA again fared better for LLC hit ratio, with 67% and L1 cache hit ratio of 11%. Regarding raw performance, the SX-Aurora architecture achieved around 5,51 MFLOPs and the P100 architecture around 2,94 MFLOPS.

We can show that, besides the advantage of higher memory bandwidth, the auto-vectorization provided by NEC's compiler is key to its success. Our primary evidence is that the vectorization vector length increases with each step in the input size. Even though the overall vectorization percentage has decreased, the more considerable vector length makes up for this loss and improves performance further.

6 CONCLUSIONS AND FUTURE WORK

Performance accelerator devices such as GPUs, are essential components for the high performance of modern supercomputers. Under this motivation, there are significant efforts in developing vector processors to also act as performance accelerating devices. In addition to having a high processing capacity SIMD, vector accelerators stand out for allowing applications to benefit from performance gains with little or no intervention in the application's source code, the fact that is less frequent in acceleration with GPUs.

In this dissertation, we did a performance review of the newly released SX-Aurora TSUBASA (KOMATSU et al., 2018; YOKOKAWA et al., 2020) vector accelerator. This analysis was done in three parts.

For the first part, we use NAS, which is a notorious benchmark of parallel applications, as well as a real application of seismic migration, called RTM. Our main objective was to verify how much performance can be gained with the SX-Aurora TSUBASA in two situations, namely: (i) without any intervention in the source code and only with the automatic optimizations of the SX-Aurora TSUBASA compiler and (ii) with simple optimization techniques known in the literature, in situations where the performance gain was not automatic.

Furthermore, Quantum walk simulation is an essential tool for studying Quantum Computing. So its performance must be perfectly attuned to the desired architecture. It is true, especially for the HiperWalk simulator and vector architectures, due to the way the simulator is designed, utilizing vector-vector and matrix-vector calculations. The second part of this work utilizes the SX-Aurora to try and harvest its potential on the quantum walk software Hiperwalk. We show that it is possible to utilize its strengths, improving general performance. However, it is necessary to consider the advantages of this kind of architecture and the specific positives the SX-Aurora has. Our primary objective in this second part is to analyze the possible improvements SX-Aurora architecture can bring to a heavily vectorizable code.

Lastly, modeling CSEM data is an arduous process. Large amounts of complex data need to be read, transferred, modified and iterated. It poses a significant challenge for developers and users. Therefore it is crucial to have the right tool for the right job. There are significant efforts in developing vector processors to act also as performance accelerator devices. In addition to having a high SIMD processing capacity, vector accelerators stand out for allowing applications to benefit from performance gains with little or no intervention in the application's source code, which is less frequent when accelerating with other heterogeneous architectures.

The third and final part of this work utilizes the SIMD, as mentioned earlier, architecture to try and harvest its potential on a CSEM modeling software called MARE2DEM.

An extensive experimental load was carried out, which showed evidence of the strengths of the vector architecture mentioned. In the first part, we were able to improve the performance (FLOPS) of the actual application RTM up to $1.9\times$ and up to $7.8\times$ for the benchmark NAS. Also, for the second part, we improved the performance, the runtime in seconds, of a real-world CSEM data set from around 2964s to 2341s, approximately 27%. For the third part, after refactoring the code base, it was possible to perform experiments to analyze NEC's improvements compared to another architecture. We achieved a performance gain of around 75% for the last use case, signifying that wider input sizes favor NEC's architecture.

We show that it is possible to utilize its strengths, improving general performance. However, it is necessary to consider the advantages of this kind of architecture and the specific positives the SX-Aurora has.

6.1 Future work

As future work, for both code adaptations, we will extend the analysis to more real-world data sets, to gauge its impacts. Implementation of performance optimization techniques such as loop unrolling and inlining, similar to Félix's (MICHELS et al., 2020), the first part of this dissertation, implementation and expand it further with loop tiling and loop interchange optimizations. These techniques were proven, in the first part of this work, to be advantageous to obtain performance gains in the SX-Aurora TSUBASA vector accelerator (MICHELS et al., 2020).

For the first part, expand our analysis to more real applications, notably Machine Learning and Artificial Intelligence applications, as well as verify if more performance optimization techniques, such as loop tiling and loop interchange can also be advantageous to obtain performance gains in the SX-Aurora TSUBASA vector accelerator.

Regarding the second part, a more in-depth study of the correlation between group refinement configurations and overall performance is desired for MARE2DEM.

Finally for the Hiperwalk code adaptation, we also expect to extend the Neblina-core development, producing a new module with CUDA to assess NVIDIA's GPUs better.

Also, the main development team behind Hiperwalk is working in decreasing the dependence in OpenCL, due to its declining support. Futhermore, there is also effort to implement the remaining planned quantum walks, namely, Szegedy's Quantum Walk, Continuous Time Quantum Walk (CTQW) and Stepped Quantum Walk.

6.2 Publications

Through the development of this dissertation three main articles were published, each one referring to one part of this dissertation.

The first published work is "*Otimização de Aplicações Paralelas em Aceleradores Vetoriais NEC SX-Aurora*" (MICHELS et al., 2020), regarding the use of optimization techniques for NEC's SX-Aurora in the NAS benchmark and RTM applications.

The second major publication regards the adaptation and analysis of a Quantum Walk simulator entitled Hiperwalk, entitled "*Simulando Passeios Quânticos em Processadores Vetoriais*" (MICHELS et al., 2022).

The third scientific article, and most recent, focuses on adapting MARE2DEM's code, and subsequently analyze its performance, to NEC's SX-Aurora. This article is entitle "Investigating Oil and Gas CSEM application on vector architectures" (MICHELS; SCHNORR; NAVAUX, 2022).

REFERENCES

ABAL, G. et al. Spatial quantum search in a triangular network. **Mathematical Structures in Computer Science**, Cambridge University Press (CUP), v. 22, n. 3, p. 521–531, Feb 2012. ISSN 1469-8072. Available from Internet: <http://dx.doi.org/10.1017/S0960129511000600>.

ABAL, G. et al. Spatial search on a honeycomb network. **Mathematical Structures in Computer Science**, Cambridge University Press (CUP), v. 20, n. 6, p. 999–1009, Nov 2010. ISSN 1469-8072. Available from Internet: <http://dx.doi.org/10.1017/S0960129510000332>.

AHARONOV, Y.; DAVIDOVICH, L.; ZAGURY, N. Quantum random walks. **Phys. Rev. A**, American Physical Society, v. 48, p. 1687–1690, Aug 1993. Available from Internet: <https://link.aps.org/doi/10.1103/PhysRevA.48.1687>.

AMBAINIS, A.; KEMPE, J.; RIVOSH, A. Coins make quantum walks faster. arXiv, 2004. Available from Internet: <https://arxiv.org/abs/quant-ph/0402107>.

ANGLéS-CASTILLO, A.; PéREZ, A. **A quantum walk simulation of extra dimensions with warped geometry**. arXiv, 2021. Available from Internet: <https://arxiv.org/abs/2105.01375>.

BAILEY, D. H. et al. The nas parallel benchmarks. **The International Journal of Supercomputing Applications**, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 3, p. 63–73, 1991.

BALL, P. First quantum computer to pack 100 qubits enters crowded race. **Nature**, v. 599, n. 7886, p. 542–542, November 2021. Available from Internet: <https://ideas.repec.org/a/nat/nature/v599y2021i7886d10.1038_d41586-021-03476-5.html>.

CASTRO, M. et al. Seismic wave propagation simulations on low-power and performance-centric manycores. **Parallel Computing**, Elsevier, v. 54, p. 108–120, 2016.

CHAVE, A. D.; CONSTABLE, S. C.; EDWARDS, R. N. 12. electrical exploration methods for the seafloor. In: ____. **Electromagnetic Methods in Applied Geophysics: Volume 2, Application, Parts A and B**. [s.n.], 2012. p. 931–966. Available from Internet: <https://library.seg.org/doi/abs/10.1190/1.9781560802686.ch12>.

CHEN, H.; XIONG, B.; HAN, Y. An effective algorithm for 2d marine csem modeling in anisotropic media using a wavelet galerkin method. **Minerals**, v. 12, n. 2, 2022. ISSN 2075-163X. Available from Internet: <https://www.mdpi.com/2075-163X/12/2/124>.

CHEN, W. et al. The effect of code expanding optimizations on instruction cache design. **IEEE Transactions on Computers**, v. 42, n. 9, p. 1045–1057, 1993.

CONSTABLE, S.; SRNKA, L. An introduction to marine controlled-source electromagnetic methods for hydrocarbon exploration. **Geophysics**, v. 72, n. 2, 2007.

COOPER, R.; MACGREGO, L. Renewed interest in csem in oil and gas exploration. **GEOExPro**, v. 17-5, 2020.

EASTTOM, W. Quantum computing and cryptography. In: ____. **Modern Cryptography: Applied Mathematics for Encryption and Information Security**. Cham: Springer International Publishing, 2021. p. 385–390. ISBN 978-3-030-63115-4. Available from Internet: <https://doi.org/10.1007/978-3-030-63115-4_19>.

EZELL, S. J.; ATKINSON, R. D. The vital importance of high-performance computing to us competitiveness. **Information Technology and Innovation Foundation, April**, v. 28, 2016.

FANAVOLL, S.; GABRIELSEN, P. T.; ELLINGSRUD, S. Csem as a tool for better exploration decisions: Case studies from the barents sea, norwegian continental shelf. **Interpretation**, v. 2, n. 3, p. SH55–SH66, 2014. Available from Internet: <https://doi.org/10.1190/INT-2013-0171.1>.

FLETCHER, R. P.; DU, X.; FOWLER, P. J. Reverse time migration in tilted transversely isotropic (tti) media. **Geophysics**, Society of Exploration Geophysicists, v. 74, n. 6, p. WCA179–WCA187, 2009.

FOWLER, P. J.; DU, X.; FLETCHER, R. P. Coupled equations for reverse time migration in transversely isotropic media. **Geophysics**, Society of Exploration Geophysicists, v. 75, n. 1, p. S11–S22, 2010.

GEIMER, M. et al. The scalasca performance toolset architecture. **Concurr. Comput.: Pract. Exper.**, John Wiley and Sons Ltd., GBR, v. 22, n. 6, p. 702–719, abr. 2010. ISSN 1532-0626.

GERNDT, M.; FÜRLINGER, K.; KEREKU, E. Periscope: Advanced techniques for performance analysis. In: **PARCO**. [S.l.: s.n.], 2005.

GLOS, A.; MISZCZAK, J. A.; OSTASZEWSKI, M. Qswalk.jl: Julia package for quantum stochastic walks analysis. **Computer Physics Communications**, v. 235, p. 414–421, 2019. ISSN 0010-4655.

GRAYVER, A. V.; STREICH, R.; RITTER, O. Three-dimensional parallel distributed inversion of CSEM data using a direct forward solver. **Geophysical Journal International**, v. 193, n. 3, p. 1432–1446, 03 2013. ISSN 0956-540X. Available from Internet: <https://doi.org/10.1093/gji/ggt055>.

HEIN, B.; TANNER, G. Quantum search algorithms on a regular lattice. **Physical Review A**, American Physical Society (APS), v. 82, n. 1, Jul 2010. ISSN 1094-1622. Available from Internet: <http://dx.doi.org/10.1103/PhysRevA.82.012326>.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture: A Quantitative Approach**. 5. ed. Amsterdam: Morgan Kaufmann, 2012. ISBN 978-0-12-383872-8.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture**. [S.l.]: Cambridge: Horgan Kaufmann Publishers, 2019.

HUANG, J.; LENG, T. Generalized loop-unrolling: a method for program speedup. In: **Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No.PR00122)**. [S.l.: s.n.], 1999. p. 244–248.

IZAAC, J. A.; WANG, J. B. pyCTQW: A continuous-time quantum walk simulator on distributed memory computers. **Comput. Phys. Commun.**, v. 186, p. 81 – 92, 2015.

JACQUELIN, M.; MARCHAL, L.; ROBERT, Y. Complexity analysis and performance evaluation of matrix product on multicore architectures. In: IEEE. **2009 International Conference on Parallel Processing**. [S.l.], 2009. p. 196–203.

JAYAKODY, M. N.; MEENA, C.; PRADHAN, P. **One-dimensional discrete-time quantum walks with general coin**. arXiv, 2021. Available from Internet: <https://arxiv.org/abs/2102.07207>.

KEY, K. Mare2dem: a 2-d inversion code for controlled-source electromagnetic and magnetotelluric data. **Geophysical Journal International**, Oxford University Press, v. 207, n. 1, p. 571–588, 2016.

KEY, K.; OVALL, J. A parallel goal-oriented adaptive finite element method for 2.5-d electromagnetic modelling. **Geophysical Journal International**, v. 186, n. 1, p. 137–154, 2011. Available from Internet: <+http://dx.doi.org/10.1111/j.1365-246X.2011.05025.x>.

KNÜPFER, A. et al. The vampir performance analysis tool-set. In: **Parallel Tools Workshop**. [S.l.: s.n.], 2008.

KNÜPFER, A. et al. Score-p: A joint performance measurement run-time infrastructure for periscope,scalasca, tau, and vampir. In: BRUNST, H. et al. (Ed.). **Tools for High Performance Computing 2011**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 79–91.

KOBAYASHI, H.; KOMATSU, K. Performance evaluation of sx-aurora tsubasa and its qa-assisted application design. In: ____. [S.l.: s.n.], 2021. p. 3–20.

KOMATSU, K.; KOBAYASHI, H. Performance evaluation of sx-aurora tsubasa by using benchmark programs. In: ____. [S.l.: s.n.], 2020. p. 69–77. ISBN 978-3-030-39180-5.

KOMATSU, K. et al. Performance evaluation of a vector supercomputer sx-aurora tsubasa. In: **SC18: International Conference for High Performance Computing, Networking, Storage and Analysis**. Dallas Convention Center Arena: IEEE, 2018. p. 685–696.

KOWARSCHIK, M.; WEISS, C. An overview of cache optimization techniques and cache-aware numerical algorithms. In: **Algorithms for memory hierarchies**. [S.l.]: Springer, 2003. p. 213–232.

KSHEMKALYANI, P. A. **Vector Processors**. University of Illinois, 2012. Available from Internet: <https://www.cs.uic.edu/~ajayk/c566/VectorProcessors.pdf>.

LARA, P. C. S.; LEãO, A.; PORTUGAL, R. Simulation of quantum walks using hpc. **Journal of Computational Interdisciplinary Sciences**, v. 6, p. 21, 2017.

MARQUEZINO, F. L.; PORTUGAL, R. The QWalk simulator of quantum walks. **Comput. Phys. Commun.**, v. 179, n. 5, p. 359–369, 2008.

MATSUURA, A. Quantum computing: A scalable, systems approach. In: ____. **Proceedings of the 18th ACM International Conference on Computing Frontiers**. New York, NY, USA: Association for Computing Machinery, 2021. p. 1. ISBN 9781450384049. Available from Internet: <https://doi.org/10.1145/3457388.3460817>.

MICHELS, F. et al. Simulando passeios quânticos em processadores vetoriais. In: **Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2022. ISSN 0000-0000.

MICHELS, F.; SCHNORR, L.; NAVAUX, P. Investigating oil and gas csem application on vector architectures. In: **Computational Science and Its Applications – ICCSA 2022 Workshops**. Porto Alegre, RS, Brazil: Springer International Publishing AG, 2022.

MICHELS, F. et al. Otimização de aplicações paralelas em aceleradores vetoriais nec sx-aurora. In: **Anais do XXI Simpósio em Sistemas Computacionais de Alto Desempenho**. Porto Alegre, RS, Brasil: SBC, 2020. p. 311–322. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/wscad/article/view/14079>.

MITTAL, S.; VETTER, J. S. A survey of cpu-gpu heterogeneous computing techniques. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 47, n. 4, p. 1–35, 2015.

MYER, D.; KEY, K.; CONSTABLE, S. Marine csem of the scarborough gas field, part 2: 2d inversion. **Geophysics**, Society of Exploration Geophysicists, v. 80, p. 187–196, 2015.

NEC. **How to Use C/C++ Compiler for Vector Engine**. 2020. <https://www.hpc.nec/api/v1/forum/file/download?id=pgNh9b>. Accessed: 09/2021.

NEC. **How to Use Fortran Compiler for Vector Engine**. 2020. <https://www.hpc.nec/api/v1/forum/file/download?id=pRdhmv>. Accessed: 09/2021.

NEC. **PROGINF/FTRACE User's Guide**. 2020. <https://www.hpc.nec/documents/sdk/pdfs/g2at03e-PROGINF_FTRACE_User_Guide_en.pdf>. Accessed: 09/2021.

NEC. **SX-Aurora TSUBASA A100-1 series user's guide**. 2020. <https://www.hpc.nec/documents/guide/pdfs/A100-1_series_users_guide.pdf>. Accessed: 09/2021.

NVIDIA. **NVIDIA Tesla P100 GPU Accelerator**. 2016. <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-p100/pdf/nvidia-tesla-p100-datasheet.pdf>. Accessed: 12-2021.

PANAHIYAN, S.; FRITZSCHE, S. Simulation of the multiphase configuration and phase transitions with quantum walks utilizing a step-dependent coin. **Phys. Rev. A**, American Physical Society, v. 100, p. 062115, Dec 2019. Available from Internet: <https://link.aps.org/doi/10.1103/PhysRevA.100.062115>.

PEREZ, A. F. et al. **Lower Numerical Precision Deep Learning Inference and Training**. 2018. <https://software.intel.com/content/www/us/en/develop/articles/lower-numerical-precision-deep-learning-inference-and-training.html>. Accessed: 09-2021.

PRICE, A. et al. 1d, 2d and 3d modeling and inversion of 3d csem data offshore west africa. In: ____. **SEG Technical Program Expanded Abstracts 2008**. [s.n.], 2008. p. 639–643. Available from Internet: <https://library.seg.org/doi/abs/10.1190/1.3063732>.

ROCHA, R. C. O. et al. Loop rolling for code size reduction. In: **2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)**. [S.l.: s.n.], 2022. p. 217–229.

SERPA, M. S. et al. Strategies to improve the performance of a geophysics model for different manycore systems. In: IEEE. **2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)**. [S.l.], 2017. p. 49–54.

SHENDE, S. S.; MALONY, A. D. The tau parallel performance system. **Int. J. High Perform. Comput. Appl.**, Sage Publications, Inc., USA, v. 20, n. 2, p. 287–311, may 2006. ISSN 1094-3420. Available from Internet: <https://doi.org/10.1177/1094342006064482>.

SHENVI, N.; KEMPE, J.; WHALEY, K. B. Quantum random-walk search algorithm. **Physical Review A**, American Physical Society (APS), v. 67, n. 5, May 2003. ISSN 1094-1622. Available from Internet: <http://dx.doi.org/10.1103/PhysRevA.67.052307>.

SILVA, S. A. da; SERPA, M. da S.; SCHEPKE, C. Técnicas de otimização loop unrolling e loop tiling em multiplicações de matrizes utilizando openmp. In: **Workshop de Iniciação Científica do WSCAD**. [S.l.: s.n.], 2016. p. 13–18.

THEODORIDIS, T.; GROSSER, T.; SU, Z. Understanding and exploiting optimal function inlining. In: **Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2022. (ASPLOS ’22), p. 977–989. ISBN 9781450392051. Available from Internet: <https://doi.org/10.1145/3503222.3507744>.

TRABESINGER, A. **Quantum computing: towards reality**. 2017. Available from Internet: <https://doi.org/10.1038/543S1a>.

TULSI, A. Faster quantum-walk algorithm for the two-dimensional spatial search. **Physical Review A**, American Physical Society (APS), v. 78, n. 1, Jul 2008. ISSN 1094-1622. Available from Internet: <http://dx.doi.org/10.1103/PhysRevA.78.012310>.

VENEGAS-ANDRACA, S. E. Quantum walks: a comprehensive review. **Quantum Information Processing**, Springer Science and Business Media LLC, v. 11, n. 5, p. 1015–1106, jul 2012. Available from Internet: <https://doi.org/10.1007%2Fs11128-012-0432-5>.

WILLSCH, M. et al. Discrete-event simulation of quantum walks. **Frontiers in Physics**, Frontiers, v. 8, p. 145, 2020.

YAVICH, N.; ZHDANOV, M. S. Finite-element em modelling on hexahedral grids with an fd solver as a pre-conditioner. **Geophysical Journal International**, Oxford University Press, v. 223, n. 2, p. 840–850, 2020.

YOKOKAWA, M. et al. I/o performance of the sx-aurora tsubasa. In: IEEE. **2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.], 2020. p. 27–35.

ZABLE, A. et al. Investigating immersive virtual reality as an educational tool for quantum computing. In: **26th ACM Symposium on Virtual Reality Software and Technology**. New York, NY, USA: Association for Computing Machinery, 2020. (VRST '20). ISBN 9781450376198. Available from Internet: <https://doi.org/10.1145/3385956.3418957>.

ZHOU, H.-W. et al. Reverse time migration: A prospect of seismic imaging methodology. **Earth-Science Reviews**, Elsevier, v. 179, p. 207–227, 2018.

APPENDIX A — RESUMO EXPANDIDO

A.1 Introdução

A computação de alto desempenho é indispensável para muitas indústrias, setores comerciais e pesquisas hoje. Ele oferece vários benefícios, desde facilitar a análise de dados até permitir novas simulações e modelagem (EZELL; ATKINSON, 2016). Assim, os avanços em HPC (High Performance Computing) são significativos. Aumentando as dimensões e diminuindo o tempo de execução, podemos beneficiar diferentes áreas da sociedade.

O desempenho do aplicativo depende em grande parte da arquitetura usada e de como o programa foi codificado para ser executado nele. Atualmente, para fins de aceleração, são utilizadas diferentes arquiteturas, como multicore, manycore, aceleradores específicos e GPUs (Graphics Processing Units).

Um grande número de arquiteturas, embora atraentes e flexíveis, impõem novos desafios ao programador (MITTAL; VETTER, 2015). A crescente complexidade da arquitetura também aumentará a dificuldade de implementação do aplicativo. Vale ressaltar a ideia de que existem vários gargalos comuns, como os encontrados no subsistema de memória, que inclui poluição *cache*, *thrashing*, entre outros.

Além de usar novas arquiteturas, várias técnicas são empregadas para aumentar o desempenho, algumas das quais são exclusivas de arquiteturas específicas. Por exemplo, a vetorização permite que uma instrução atue em vários dados, mas nem todos implementam suporte para isso. Outras otimizações incluem o aumento da taxa de acerto da memória *cache*, que é possível em diferentes arquiteturas.

Nesse sentido, a NEC Corporation lançou uma nova arquitetura, um processador vetorial chamado SX-Aurora. Este processador possui oito núcleos de processamento a 1,6 GHz e três níveis de memória cache (KOMATSU et al., 2018). Uma das vantagens desta arquitetura em relação às demais existentes é o tamanho de suas unidades vetoriais. Além disso, o compilador NEC toma decisões automaticamente. Ou seja, identifica áreas vetorizáveis e gera código para isso. No entanto, o compilador ainda precisa de ajuda do programador para facilitar a interpretação do código, além de melhorar a vetorização automática, seguindo orientações específicas e, neste caso, utilizando técnicas de otimização como unrolling e inlining de loops.

As arquiteturas vetoriais são Single Instruction Multiple Data (SIMD), que têm grande potencial para aplicações científicas altamente paralelizáveis. Entre eles estão aplicações numéricas, previsão de tempo, processamento multimídia (KSHEMKALYANI, 2012), simulação de colisão (HENNESSY; PATTERSON, 2019), compressão de dados, entre outros. Uma característica proeminente desses processadores vetoriais é a possibilidade de usar uma instrução para reproduzir centenas de operações. Além disso, todos os resultados dos elementos de um vetor são independentes e, portanto, não é necessário verificar os dados resultantes. O acesso à memória é feito apenas uma vez para cada vetor, inferindo uma pequena latência de acesso à memória.

A.1.1 Simulação

Outra forma de melhorar a produção científica é através da simulação. Os simuladores são ferramentas necessárias e poderosas utilizadas em diversos campos, desde pesquisas até empreendimentos comerciais e industriais.

A.1.1.1 Prospecção de petróleo e gás

Os investimentos da indústria petroquímica em eletromagnetismo de fonte controlada (CSEM) só aumentaram nos últimos 15 anos, e o interesse acadêmico acompanha isso (CONSTABLE; SRNKA, 2007) (COOPER; MACGREGO, 2020) (FANAVOLL; GABRIELSEN; ELLINGSRUD, 2014). A principal razão para esses investimentos é reduzir o risco. O CSEM fornece mais dados e, portanto, mais informações para fornecer novos insights sobre o fundo do mar, reduzindo consequentemente o risco.

A aquisição de dados CSEM produz uma grande quantidade de dados, levando a um tremendo problema computacional para resolver a modelagem inversa. Algumas regiões com geometria complicada podem exigir dados adicionais, principalmente para produzir inversão 3D completa. No entanto, a inversão 2D é muito mais rápida e fornece uma interpretação mais direta dos dados reais em um tempo de execução mais curto, tornando-a uma abordagem mais robusta, sensata e viável (PRICE et al., 2008).

Uma implementação necessária para gerenciar este tipo de dados CSEM é a aplicação "Modeling with Adaptively Refined Elements for 2D Electromagnetics", referido neste projeto como MARE2DEM (KEY, 2016). MARE2DEM é um código de código aberto para inversão 2D de dados CSEM, dados magnetotelúricos (MT) e dados EM de poços superficiais, por elementos finitos adaptativos paralelos para ambientes onshore, offshore e de fundo de poço. Devido à grande quantidade de dados fornecidos pelo CSEM, é necessário alto poder computacional para executar tal conjunto de dados com eficiência. Portanto, código eficiente e um computador poderoso são preferidos.

A.1.2 Simuladores de caminhada quântica

Os avanços da computação quântica inevitavelmente aumentarão o poder e a eficiência computacional (EASTTOM, 2021). Essas melhorias são necessárias à medida que os computadores modernos lutam para processar sistemas biológicos complexos, estruturas químicas e novos materiais (TRABESINGER, 2017). Mesmo áreas improváveis, como Realidade Virtual, poderão aproveitar essa nova tecnologia (ZABLE et al., 2020). A IBM alcançou 127 qubits em seu mais recente chip de computação quântica, marcando um novo marco para toda a indústria, a primeira contagem de qubits de três dígitos (BALL, 2021). A IBM pretende ultrapassar 1000 qubits até 2023, visando um aumento nunca visto no poder computacional.

No entanto, esses computadores e processadores quânticos não estão totalmente disponíveis para o público em geral. Para preencher essa lacuna entre a comunidade científica em geral e a ciência da computação quântica, os simuladores quânticos são a melhor alternativa.

Uma categoria de tais simuladores são os simuladores de caminhada quântica, embora sejam raros. Um desses casos raros é o simulador Hiperwalk (LARA; LEãO; PORTUGAL, 2017). Este simulador é dividido em três partes. A primeira é uma interface de usuário escrita em Python que gera arrays e vetores com base na entrada do usuário no programa principal. A segunda parte consiste nos cálculos para as etapas quânticas. É utilizada a biblioteca Neblina-core, desenvolvida no Laboratório Nacional de Computação Científica (LNCC). Esta biblioteca facilita cálculos do tipo matriz-vetor e vetor-vetor em arquiteturas heterogêneas. Por fim, a última parte é um módulo capaz de calcular distribuições estatísticas e gerar arquivos de dados de saída.

A.1.3 Objetivos

Esta dissertação de mestrado realizou um estudo, avaliação, adaptação e otimização de quatro aplicações distintas em três partes distintas. Esses aplicativos são o benchmark NAS, Reverse Time Migration (RTM), MARE2DEM e Hipewalk.

A primeira parte mencionada acima é otimizar o desempenho de aplicações reais e um conjunto de benchmarks utilizando técnicas de loop unrolling e inlining, buscando melhorar a vetorização automática realizada pelo compilador. Mais precisamente, o presente trabalho apresenta as seguintes contribuições:

- Uma análise de desempenho experimental do acelerador vetorial NEC SX-Aurora TSUBASA é realizada. Um benchmark de mapas sintéticos paralelos e um aplicativo real é usado.
- Mostrar que as técnicas de unrolling e inlining de loop são capazes de melhorar significativamente o desempenho das aplicações quando executadas no SX-Aurora TSUBASA em situações em que o ganho de desempenho não é automático pelo seu compilador.

A segunda parte é uma investigação, adaptação e análise da execução do MARE2DEM utilizando uma arquitetura vetorial, o SX-Aurora, e a tradicional arquitetura x86. Serão utilizados dois conjuntos de dados, um sintético e um caso real fornecido pela Petrobras, empresa brasileira de petróleo e gás. Portanto, os principais objetivos deste trabalho são:

- Descreva a implementação do MARE2DEM no SX-Aurora da NEC. A biblioteca matemática padrão da Intel foi substituída pelo MARED2DEM no SX-Aurora porque a arquitetura da NEC nãot apoiá-lo. Todas as entradas que exigiam a biblioteca matemática da Intel foram reescritas, suportando as bibliotecas matemáticas da NEC.
- Investigar a análise de desempenho desta implementação comparando-a com CPUs de arquitetura x86.

A terceira parte é referente à adaptação e análise de desempenho do Hiperwalk no SX-Aurora da NEC. Os principais objetivos são:

- Apresentar o simulador Hiperwalk, explicando seus casos de uso;
- Descrever a implementação do kernel do HiperWalk no SX-Aurora TSUBASA da NEC;
- Uma análise de desempenho de nossa implementação comparando-a com as GPUs da NVIDIA.

O objetivo principal desta dissertação, como um todo, é elevar o suposto potencial que uma arquitetura vetorial pode ter nas circunstâncias certas, especialmente o SX-

Aurora TSUBASA.

A.1.4 Contribuições deste trabalho

As principais contribuições deste trabalho são a análise de desempenho de técnicas clássicas de otimização para SX-Aurora usando duas aplicações diferentes, e a adaptação de código e análise de desempenho de um simulador de caminhada quântica (Hiperwalk) e um código de elementos finos geofísicos usados na área de gás e petróleo indústria (MARE2DEM). Verificou-se que as técnicas de otimização foram capazes de aumentar o desempenho na maioria dos casos, e as adaptações de código também tiveram melhorias de desempenho nas circunstâncias corretas.

A.1.5 Principais resultados alcançados

Este trabalho apresenta como principais resultados um ganho de desempenho nas tres partes analisadas. Na primeira parte, sendo essa relacionada as técnicas de otimização, teve um aumento de desempenho de 1,9 vezes no caso real utilizando RTM e 7,8 para o *benchmark* NAS. A segunda parte pode-se observar um aumento de aproximadamente 27% no desempenho. Já na terceira parte percebe-se um ganho de 75%.