



## ADVANCED COMPUTATIONAL STRATEGIES FOR REVERSE TIME MIGRATION

Carlos Henrique dos Santos Barbosa

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Civil, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Civil.

Orientador: Alvaro Luiz Gayoso de Azeredo  
Coutinho

Rio de Janeiro  
Março de 2023

ADVANCED COMPUTATIONAL STRATEGIES FOR REVERSE TIME MIGRATION

Carlos Henrique dos Santos Barbosa

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA CIVIL.

Orientador: Alvaro Luiz Gayoso de Azeredo Coutinho

Aprovada por: Prof. Alvaro Luiz Gayoso de Azeredo Coutinho

Prof. Webe João Mansur

Prof. José Luis Drummond Alves

Prof. Fernando Alves Rochinha

Prof. Philippe Olivier Alexandre Navaux

Dr. Djalma Manoel Soares Filho

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2023

Barbosa, Carlos Henrique dos Santos

Advanced Computational Strategies for Reverse Time Migration/Carlos Henrique dos Santos Barbosa. – Rio de Janeiro: UFRJ/COPPE, 2023.

XVI, 116 p. 29, 7cm.

Orientador: Alvaro Luiz Gayoso de Azeredo Coutinho

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Civil, 2023.

Referências Bibliográficas: p. 82 – 94.

1. Seismic Imaging. 2. High-performance Computing. 3. Reverse Time Migration. 4. Finite Difference Method. 5. Computational Optimization and Parallelization Strategies. I. Coutinho, Alvaro Luiz Gayoso de Azeredo. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Civil. III. Título.

*"No que diz respeito ao empenho,  
ao compromisso, ao esforço, à  
dedicação, não existe meio-termo.  
Ou você faz uma coisa bem-feita  
ou não faz." - Ayrton Senna*

# Agradecimentos

Primeiramente, gostaria de agradecer a Deus pelas oportunidades proporcionadas, tanto em minha vida pessoal quanto profissional. Experiências estas que me transformaram no ser humano que sou hoje. Além disso, me deu como presente uma família maravilhosa que me apoia e incentiva incansavelmente. Com isso, quero expressar e registrar a eterna gratidão que tenho à minha mãe Rosilda e ao meu pai Altair por me apoiarem em todos os momentos da vida. Nunca vou esquecer dos momentos de dificuldades que passamos, mas mesmo assim me incentivaram a buscar meus objetivos.

A família Santos e a família Barbosa quero agradecer por sempre estarem torcendo e vibrando com cada conquista obtida, mesmo sendo pequena. Aos momentos festivos que não consegui estar presente devido a compromissos, digo que senti muita falta de estar reunido com todos.

À minha amada esposa Adna Vasconcelos que me acompanha dia após dia, sou grato a todo carinho e a amizade. Sua presença me fortalece na busca incessante por felicidade, confidencialidade e qualidade de vida para que possamos aproveitar ao máximo todo o momento juntos.

Aos meus amigos agradeço por todo apoio durante minha jornada até o momento. Em especial quero agradecer aos amigos Bruno de Souza, Luana Nobre, Márcio Sampaio, e Mariane Rita. São amigos que levarei para a eternidade. Com todos compartilhei momentos muito importantes que me permitiram crescer como pessoa.

Ao professor Alvaro Coutinho agradeço pela oportunidade de trabalharmos juntos. Com este profissional de alto nível tive o privilégio de aprender muito.

Por fim, gostaria de agradecer às agências de fomento, FAPERJ e CNPq, e a Petrosbras pelo apoio financeiro da pesquisa. Além disso, agradecer a disponibilização de recursos computacionais nos supercomputadores Santos Dumont do Laboratório Nacional de Computação Científica (LNCC - Petrópolis), Lobo Carneiro da Universidade Federal do Rio de Janeiro e no supercomputador vetorial SX-Aurora TSUBASA da NEC Latin AMERICA S.A..

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

## ESTRATÉGIAS COMPUTACIONAIS AVANÇADAS PARA MIGRAÇÃO REVERSA NO TEMPO

Carlos Henrique dos Santos Barbosa

Março/2023

Orientador: Alvaro Luiz Gayoso de Azeredo Coutinho

Programa: Engenharia Civil

Esta tese apresenta avanços algorítmicos para a técnica de Migração Reversa no Tempo (MRT) e aplicações geofísicas baseadas na equação completa da onda. Tais avanços estão relacionados à introdução de métodos de decimação, compressão e reconstrução do campo de ondas. A inserção de tais métodos tem como objetivo a redução da demanda de armazenamento em disco normalmente exigida pelos algoritmos clássicos de MRT e também, em alguns casos, o tempo de processamento. Além disso, as metodologias de compressão, decimação e reconstrução do campo de ondas são estendidas para a MRT com o método de amostragem Monte Carlo (MC-MRT). A MC-MRT é utilizada para quantificar incertezas de grandezas físicas/geológicas de interesse, tais como variações de amplitude e deslocamento espacial dos refletores sísmicos. Nossa experiência sugere que os avanços algorítmicos podem ser estendidos para qualquer técnica baseada na equação completa da onda, tais como a *Full Waveform Inversion*, MRT por Mínimos Quadrados e *Full Waveform Imaging*. Mais que isso, os avanços algorítmicos se beneficiam das novas tecnologias com múltiplas CPUs, GPUs e/ou processadores vetoriais (PV). Análises do tempo de execução, demanda de armazenamento em disco, *overhead* e *speedup* com base nas ferramentas de perfilagem apontam para a redução da demanda computacional, bem como, expressiva utilização das GPUs e PVs. Estes fatos podem ser observados nas altas taxas de vetorização e utilização da GPU/PV, e também nos baixos valores de *cache missing*. Os resultados demonstram a viabilidade do desenvolvimento de metodologias de redução de armazenamento em disco e suas respectivas implementações em tecnologias de *hardware* avançadas para alcançar melhor utilização dos recursos computacionais. O desenvolvimento dessas metodologias dão suporte à MRT para alcançar as soluções acuradas em um tempo computacional mais hábil com mínima influência nos resultados finais, mesmo para os métodos iterativos, tal como a quantificação de incertezas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

## ADVANCED COMPUTATIONAL STRATEGIES FOR REVERSE TIME MIGRATION

Carlos Henrique dos Santos Barbosa

March/2023

Advisor: Alvaro Luiz Gayoso de Azeredo Coutinho

Department: Civil Engineering

This thesis presents algorithm advances for the Reverse Time Migration (RTM) technique and geophysical applications based on two-way wave equations. The advances are related to the introduction of decimation, compression, and wavefield reconstruction methods. Inserting such methods aims to reduce persistent storage usually required by RTM algorithms and, in some cases, processing time reduction. Also, the decimation, compression, and wavefield reconstruction methodologies are extended to an RTM wrapped into the Monte Carlo (MC-RTM) sampling method. MC-RTM can be used to quantify uncertainties of physical/geological quantities of interest, such as amplitude variations and/or seismic reflector displacements. Our experiences suggest that the algorithm advances can also be extended to others techniques based on the two-way wave equation like the Full Waveform Inversion, Least-Squares RTM, and Full Waveform Imaging. Furthermore, the RTM algorithm advances are beneficial for emerging heterogeneous machines based on multi-CPU, multi-GPU, and multi-VEs (vector engines) technologies. Analysis of the execution time, storage demand, overhead, and speedup based on profiling tools indicates a reduction of the computational requirements and expressive computation device usage. These facts can be observed by looking at the high vectorization, GPU usage ratios, and extremely low cache miss values. The results demonstrate that it is viable to develop storage reduction methodologies and implement them on advanced machines to achieve better use of the computational resources. These methodologies support the RTM technique aiming to obtain fast solutions without hampering the outcomes, even for many-query problems such as uncertainty quantification.

# Contents

<b>Contents</b>	<b>viii</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiv</b>
<b>List of Algorithms</b>	<b>xvi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Objectives . . . . .	3
1.2 Outline . . . . .	5
1.3 Contributions and Published Materials . . . . .	6
<b>2 Seismic Imaging</b>	<b>7</b>
2.1 Generalized Full Waveform Inversion . . . . .	8
2.1.1 Reverse Time Migration . . . . .	10
<b>3 Computational Strategies to Boost Reverse Time Migration</b>	<b>12</b>
3.1 High-order finite difference method for wavefield propagation . . . . .	12
3.1.1 Numerical implementation of acoustic wave equation . . . . .	16
3.1.2 Computational cost of high-order discretizations . . . . .	17
3.2 Data compression for wavefield storage . . . . .	24
3.3 Wavefield reconstruction . . . . .	27
<b>4 Computational Implementation</b>	<b>32</b>
4.1 Optimized multi-core baseline implementation . . . . .	32
4.2 Vector processor parallelism implementation . . . . .	35
4.3 GPU parallelism implementation . . . . .	35
4.4 Hybrid parallelism implementation . . . . .	41
<b>5 Numerical Experiments</b>	<b>43</b>
5.1 Computational performance analysis . . . . .	43
5.1.1 Seismic modeling experiment . . . . .	44

5.1.2	Seismic migration experiment . . . . .	48
5.2	Data compression . . . . .	54
5.2.1	Seismogram compression . . . . .	55
5.2.2	Wavefield compression . . . . .	60
5.3	Uncertainty quantification application . . . . .	66
5.3.1	Reverse time migration under uncertainty . . . . .	68
5.3.2	Workflow computational performance analysis . . . . .	70
5.3.2.1	General computational aspects . . . . .	71
5.3.2.2	Computational performance analysis . . . . .	71
5.3.2.3	Hybrid implementation performance comparisons . . . . .	73
5.3.3	Uncertainty quantification analysis . . . . .	74
<b>6</b>	<b>Final Remarks and Perspectives</b>	<b>78</b>
	<b>References</b>	<b>82</b>
	<b>Appendices</b>	<b>95</b>
	<b>Appendix A Boundary Conditions</b>	<b>96</b>
A.1	Non-reflecting boundary condition . . . . .	96
A.2	Convolutional perfectly matched layer . . . . .	97
A.2.1	Application to isotropic acoustic wave equation . . . . .	97
A.3	Random boundary condition . . . . .	99
	<b>Appendix B Wavefield Decimation</b>	<b>100</b>
	<b>Appendix C Source and Reconstructed Wavefield Equivalence</b>	<b>102</b>
	<b>Appendix D Vector Processing</b>	<b>104</b>
D.1	Basic architecture of NEC SX-Aurora TSUBASA . . . . .	105
	<b>Appendix E The basics of OpenACC</b>	<b>107</b>
E.1	Compute directives . . . . .	108
E.2	Data management directives . . . . .	108
E.3	Data handling clauses . . . . .	109
E.4	Control flow clauses . . . . .	109
	<b>Appendix F Workflow main building blocks</b>	<b>110</b>
F.1	Estimating uncertain velocity fields . . . . .	110
F.2	Assessing uncertainty in images . . . . .	112

<b>Appendix G Profiling Reports</b>	<b>114</b>
G.1 Profiling analysis on SX-Aurora TSUBASA . . . . .	114
G.2 Profiling analysis on NVIDIA GPU . . . . .	116

# List of Figures

3.1	Propagation of the wavefield in the constant 3D velocity field of 3000.0 m/s for the instants 0.6 s (upper), 1.0 s (middle), and 1.5 s (lower). . . . .	18
3.2	Seismic signal for the discrete wave equation that is 2nd-order accurate in time and space. The upper image considers a grid with $801 \times 801 \times 361$ points, where the grid space is 12.5 meters, and the lower image considers a grid with $401 \times 401 \times 181$ points, where the grid space is 25.0 meters. . . . .	21
3.3	Seismic signal for the discrete wave equation that is 2nd-order accurate in time and 4th-order accurate in space. The upper image considers a grid with $801 \times 801 \times 361$ points, where the grid space is 12.5 meters, and the lower image considers a grid with $401 \times 401 \times 181$ points, where the grid space is 25.0 meters. . . . .	22
3.4	Seismic signal for the discrete wave equation that is 2nd-order accurate in time and 8th-order accurate in space. The upper image considers a grid with $801 \times 801 \times 361$ points, where the grid space is 12.5 meters, and the lower image considers a grid with $401 \times 401 \times 181$ points, where the grid space is 25.0 meters. . . . .	23
3.5	The left column shows the representation of the source wavefield for the instants 0.25s (A), 1.5s (B), 2.5s (C), and 3.0s (D). The right column shows the representation of the wavefield reconstruction of the source wavefield for instants 0.25s (E), 1.5s (F), 2.5s (G), and 3.0s (H). The results were provided by Algorithm 4. . . . .	31
5.1	3D velocity field provided by the HPC4E Seismic Test Suite. . . . .	44
5.2	Seismic modeling speedup across the platforms Santos Dumont CPU Cluster, NVIDIA Volta V100 and SX-Aurora TSUBASA Vector Engine for the $501 \times 501 \times 235$ grid. . . . .	46
5.3	Seismic modeling speedup across the platforms Santos Dumont CPU Cluster, NVIDIA Volta V100 and SX-Aurora TSUBASA Vector Engine for the $901 \times 901 \times 416$ grid. . . . .	46
5.4	Percentage time spent on GPU calculations and data transfer for the seismic modeling. . . . .	47

5.5	Propagation of the wavefield in the 3D velocity field provided by the HPC4E Seismic Test Suite for the instants 0.6 s (upper), 1.5 s (middle), 2.0 s (lower). The results were provided by Algorithm 1. . . . .	49
5.6	Reverse Time Migration speedup across the platforms Santos Dumont CPU Cluster, NVIDIA V100 and SX-Aurora TSUBASA Vector Engine for the $501 \times 501 \times 235$ grid. . . . .	51
5.7	Percentage time spent on the VE Aurora TSUBASA to generate the random boundary, solve the wave equation, and calculate the convolutional imaging condition for the RTM based on the source wavefield reconstruction. . . . .	52
5.8	Percentage time spent on GPU calculations and data transfer for the RTM based on the source wavefield reconstruction. . . . .	52
5.9	Seismic image for one migrated shot of 3D HPC4E Seismic Test Suite. . . . .	53
5.10	Seismic image for the 3D HPC4E Seismic Test Suite. . . . .	54
5.11	2D velocity field from the Marmousi2 benchmark. . . . .	55
5.12	RTM outcome after migrating the reference seismograms (upper image). Figures lower left and lower right represent the 2D Fourier spectra of the white box regions shown in the migrated seismic image. . . . .	57
5.13	RTM outcome after migrating the lossless compressed seismograms (upper image). Figures lower left and lower right represent the 2D Fourier spectra of the white box regions shown in the migrated seismic image. . . . .	57
5.14	RTM outcome after migrating the lossy compressed seismograms (upper image). Figures lower left and lower right represent the 2D Fourier spectra of the white box regions shown in the migrated seismic image. . . . .	58
5.15	2D velocity field from the Marmousi benchmark. . . . .	61
5.16	RTM outcomes after migration using the forward-propagated source wavefield (a) without compression, (b) with lossless compression, (c) with a lossy tolerance compression of $10^{-6}$ , and (c) with a lossy tolerance compression of $10^{-4}$ . . . . .	62
5.17	Difference among the reference migrated image (no compression) and (a) the migrated image after applying lossless compression on the source wavefield, (b) the migrated image after applying a lossy tolerance compression of $10^{-6}$ on the source wavefield, and the migrated image after applying a lossy tolerance compression of $10^{-4}$ on the source wavefield. . . . .	63
5.18	Size per time slice for the 2D source wavefield without compression (blue line), with lossless compression (orange line), with a lossy tolerance compression of $10^{-4}$ (green line), and with a lossy tolerance compression of $10^{-6}$ (red line). . . . .	64

5.19	Size per time slice for the 3D source wavefield without compression (blue line), with lossless compression (orange line), with a lossy tolerance compression of $10^{-4}$ (green line), and with a lossy tolerance compression of $10^{-6}$ (red line).	65
5.20	MC-RTM speedup across the platforms Santos Dumont CPU Cluster and NVIDIA V100.	73
5.21	Figures (a) and (b) show the mean and confidence index of the Marmousi2 velocity fields.	75
5.22	Figures (a) and (b) show the mean and confidence index of the migrated seismic images.	76
5.23	Comparison between mean migrated seismic image built by the RTM with storage (a) and the RTM with wavefield reconstruction (b). Figure (c) shows relative error between seismic images (a) and (b).	77
G.1	Performance information for the seismic modeling using the <code>ftrace</code> tool on SX-Aurora TSUBASA vector processor.	114
G.2	Performance information for the RTM using the <code>ftrace</code> tool on SX-Aurora TSUBASA vector processor.	115
G.3	Seismic modeling profiling data of CUDA-related activities on both CPU and GPU using the <code>nvprof</code> tool.	116
G.4	RTM profiling data of CUDA-related activities on both CPU and GPU using the <code>nvprof</code> tool.	116

# List of Tables

3.1	Finite-difference coefficients for the second derivative discretization. . . .	14
3.2	Execution time of seismic modeling for three FD-order discretizations on two different grid sizes. The blue execution times indicate the wave equation solution without numerical dispersion. . . . .	20
5.1	Seismic Modeling performance measurements for different grid sizes on the NVIDIA V100 and SX-Aurora TSUBASA. The average time is calculated for ten execution time measurements. . . . .	45
5.2	Comparison of hard disk and time requirements for the 3D RTM implementation with the wavefield storage, and the wavefield reconstruction. .	50
5.3	Comparison of the total execution time for the 3D RTM implementation with wavefield reconstruction on CPU cluster and NVIDIA V100. . . . .	53
5.4	Seismograms fixed accuracy lossy and lossless compression comparison. The error tolerance for lossy compression is set to $10^{-4}$ . The values represent the minimum and maximum compression among the 160 seismograms.	58
5.5	Seismograms fixed accuracy lossy and lossless compression comparison. The error tolerance for lossy compression is set to $10^{-4}$ . The values represent the minimum and maximum compression among the 1681 seismograms for the 3D HPC4E benchmark. . . . .	59
5.6	Time requirements and overhead for the 3D RTM implementation considering different compression rates for the seismograms. The error tolerance for lossy compression is set to $10^{-4}$ . The measurements take into account ten executions of each implementation. . . . .	60
5.7	Time requirements and overhead for the 3D RTM implementation considering different compression rates for the source wavefield. The error tolerance for lossy compressions are set to $10^{-6}$ and $10^{-4}$ . The measurements take into account ten executions of each implementation. . . . .	66
5.8	Comparison of hard disk and time requirements for the MC-RTM implementation with the wavefield storage and reconstruction methods. . . . .	72

5.9 Comparison of the total execution time for the 2D MC-RTM implementation based on wavefield storage and wavefield reconstruction on CPU and NVIDIA V100 clusters, respectively. . . . . 74

# List of Algorithms

1	Seismic modeling for the wavefield propagation . . . . .	17
2	Reverse Time Migration . . . . .	26
3	Reverse Time Migration with Wavefield Compression . . . . .	28
4	Reverse Time Migration with Wavefield Reconstruction . . . . .	30
5	Seismic Modeling GPU Implementation . . . . .	36
6	RTM GPU Implementation based on Algorithm 2 . . . . .	39
7	RTM GPU Implementation based on Algorithm 4 . . . . .	40
8	Reverse Time Migration with Compressed Seismograms . . . . .	56
9	Workflow for seismic imaging with quantified uncertainty . . . . .	67
10	Monte Carlo Reverse Time Migration for uncertainty analysis. . . . .	69
11	Improved Monte Carlo Reverse Time Migration for uncertainty analysis. . . . .	70
12	Decimation . . . . .	100
13	Wavefield decimation . . . . .	101

# Chapter 1

## Introduction

Mapping subsurface geological structures can be done by applying geophysical methods which are related to the physical properties of the medium. For instance, the seismic method consists of measuring reflected/refracted seismic waves and from them estimating the density and elastic moduli, which determine the propagation velocity of seismic waves [62, 103]. The seismic method has wide applicability in the main fields of exploration for fossil fuels, exploration for mineral deposits, and engineering/construction site investigation. Particularly, seismic methods for the exploration of hydrocarbons are the most important technique due to their routine, widespread, and amount of money expended annually [62].

In this context, Reverse Time Migration (RTM) is a depth seismic migration technique that provides a reliable high-resolution representation of the Earth's subsurface useful for the exploration of hydrocarbons [135]. Basically, the classical RTM is based on the two-way wave equation and an imaging condition [30]. However, RTM can also be formulated as an iterative inversion scheme, where the inversion process leads to the called Least-squares Reverse Time Migration (LSRTM) and Full Waveform Inversion (FWI). LSRTM assumes the background parameter model invariant over the iterations, which is different from FWI that updates the parameter model after each iteration [59, 60, 103].

Independent of how it is formulated, RTM requires numerical methods to solve the two-way wave equation and strategies to deal with truncated domains and the forward-propagated wavefield (or source wavefield) to build the imaging condition. Generally, the two-way wave equation is solved by numerical methods such as the Finite Difference Method (FDM) and the Finite Element Method (FEM) [103]. Perfectly Matched Layer (PML) [13], Convolutional PML (CPML) [63, 97], and non-reflecting boundary conditions based on damping factors [25] are the main methods used to avoid artificial reflections coming from the boundaries due domain truncation of the Earth subsurface. Last but not least, a natural approach to deal with the source wavefield needed to build imaging conditions is to store it on memory or disk, depending on the storage requirements.

RTM is classified as a time-consuming and data-intensive technique [9, 100, 105],

and advances in seismic wave propagation algorithms, wavefield storage, and hardware acceleration are some of the main challenges concerning RTM [135]. From a time-consuming viewpoint, RTM algorithms need to be developed to take full advantage of all computer hardware technologies. Among them, we refer to Central Processing Units (CPUs) equipped with multi-cores [9, 61, 101, 104], graphic processing units (GPUs) [8, 100, 104, 105], and vector processors (VPs) [10, 43, 87, 89, 90] machines to boost its computational performance. Multi-CPU implementations are mainly directed toward vectorization, load balancing, and cache memory usage [9, 104] as well spatial and temporal data reuse known as tiling [1, 101, 104]. Emerging heterogeneous architectures such as CPU-GPU machines demand hybrid implementations on single and multiple nodes using, for instance, combined technologies like Message Passing Interface (MPI), Open Multi-Processing (OpenMP) and Compute Unified Device Architecture (CUDA) or Open Accelerators (OpenACC) libraries [8, 100, 105]. Hybrid implementations may produce hard-to-read codes if not used carefully. In this sense, it is common to find research aiming at developing portable codes for heterogeneous computational platforms focused on preserving performance and efficiency [8, 100, 101, 105]. Another powerful machine for solving problems in a wide variety of scientific research fields is the super-computer based on vector processors (VPs). Recently, NEC company introduced to the public the SX-Aurora TSUBASA which is a vector machine that has two major advantages over its predecessors. First of all, there is no need to instruct data exchange between the host and co-processors because the application is executed mostly on the co-processors and data transfer of system calls is, in many cases, negligible. Because of that, the second major advantage is the possibility of writing a program entirely in a standard programming language like C/C++ and FORTRAN [3, 42, 126].

The source wavefield is another bottleneck concerning RTM due to the amount of information that has to be stored on a disk to build the imaging condition [135]. Strategies to diminish input/output (I/O) related to the source wavefield revolve around mainly of full/partial storage or reconstruction techniques [9, 31, 32, 44, 72, 93, 112, 113]. Methods for full/partial storing the source wavefield can take advantage of Nyquist sampling theory to decimate the temporal evolution of the wavefield propagation [32, 93, 112]. This process can be thought of as the simplest way to compress the source wavefield [4]. Lossless/lossy compressors linked or not to decimation approaches can also reduce even more the storage demand. If desired, using carefully lossy compressors do not hamper the final seismic images [9, 112]. Instead of storing the wavefield, its reconstruction is a viable possibility. This can be done by checkpoint methods [44, 113], using wavefield recording around the boundary [31, 93], or by initial value reconstruction (IVR) [32, 72, 93]. The basic concept of these methods is to use stored information of the wavefield solutions on the boundary, the last wavefield moments, or decimated wavefields that not respects the Nyquist sampling theory. The reconstruction can be done by introducing new equations to

the problem able to recover the missing information [32, 44, 93, 113]. An interesting third option that does not fall into the mentioned classes is the dynamic approach which delimits the computational domain of the seismic wavefront. This approach leads to memory savings and reduced processing times at once compared to the conventional RTM which stores the full wavefield [95].

Generally, introducing strategies to reduce data transfer between the main memory and disk and maximize CPU or GPU usage can result in increasing the execution time of geophysical applications. Opposite to that, NOGUEIRA and PORSANI [95] show that it is possible to save memory and reduce the processing time of the RTM at once through their wavefront dynamic approach. However, the authors do not make themselves clear about the platform they have used and if their approach keeps the performance in other hardware. In this context, developments of the 2D RTM considering a random boundary condition (RBC) and ways to attenuate artificial events reflected at the boundaries produce reliable results and can be beneficial for CPU/GPU-based RTM implementations in terms of memory savings and execution time [79]. The RBC method [32] explores low correlations with non-coherent signals coming from an artificial boundary with random velocities instead of suppressing unwanted waves by inserting new equations into the problem like the absorbing boundary conditions such as CPML and damping factors. Due to its features, the RBC is mainly used in migration algorithms because the stacking process attenuates the non-coherent wavefield [32, 79, 109].

Defining a set of strategies that balances execution time and storage demand on emerging heterogeneous platforms is essential for methods based on the two-way wave equation. As mentioned earlier, RTM shares many features with the LS-RTM and FWI techniques. In this sense, every advance in the RTM can be transferred to LS-RTM and FWI, and this is vital because their computational cost grows with the number of iterations and frequencies (for the multi-scale FWI case) [119, 124]. Such developments can become more important for probabilistic scenarios used to quantify uncertainties of physical properties. In this case, the number of iterations for the RTM or FWI can reach the order of hundreds or even thousands depending on the parameter dimensionality [10, 47, 106, 129, 133, 134].

## 1.1 Objectives

The motivation of this work relies on the development of algorithmic strategies for RTM that allows efficient and portable computational implementations among heterogeneous supercomputers. RTM method is used to provide a representation of the Earth's subsurface named seismic image. To reach that, RTM has two major components, which are a wave equation and an imaging condition [30]. The first one consists of partial differential equations (PDEs), which describe the wave propagation through heteroge-

neous media. On the other hand, seismic imaging builds seismic images with solutions of the wave equation under different initial and boundary conditions. The solutions refer to the forward-propagated wavefield (source wavefield) and backward-propagated wavefield (receiver wavefield) [103, 135].

Two major problems related to algorithm developments of RTM consist of defining strategies to deal with a huge amount of information demanded by the source wavefield solution and providing efficient and portable codes for the main available platforms in the academy and industry. The source wavefield has to be accessed backward in time during the receiver wavefield calculation to build the imaging condition. Such requirement forces the source wavefield storage, which is already constantly exchanged between CPU and memory, for example, to update the solution over space and time. Such features characterize the RTM as memory-bounded [82, 83]. Moreover, solving the governing wave equations of RTM by numerical methods is computationally intensive due to the number of operations to update a single cell<sup>1</sup>. For instance, using the FDM to discretize the two-way isotropic acoustic wave equation demands a different number of operations per grid point, and this number depends on the discretization order. Because of that, RTM is also classified as a compute-bounded technique [82, 83]. Approaches to handling the memory and compute-bound characteristics are quite different and need to be treated carefully.

In this context, we provide algorithmic advances for the RTM technology that are beneficial for the emerging heterogeneous machines based on multi-CPU, multi-GPU, and multi-VP. We describe and analyze RTM algorithms that take into account decimation, compression, and wavefield reconstruction methods. In this sense, before storing the source wavefield, RTM compresses it after its decimation based on the Nyquist theory. The strategy reduces the persistent storage required by the method. Besides, compressing the seismogram incomes has also its importance due to the amount of information generated on seismic surveys. Hence, it is vital to understand the impact of using compressed seismograms on the migration process that builds the seismic image. We analyze how much is possible to reduce storage and the quality of the seismic images in terms of the RMS error between them and also between their frequency per wavenumber (f-k) spectrum. Considering the wavefield reconstruction method, we introduce a third wave equation to the RTM that allows the full reconstruction of the source wavefield. This is possible because we adjust the initial and boundary conditions to keep the wavefield inside the domain. Normally, such strategy generates reflections coming from the boundaries blurring the final seismic imaging. To overcome this issue, we use the RBC first introduced by CLAPP [32] to randomize the wavefield on the boundary aiming to explore the attenuation of the non-coherent signal in the stacking process [32, 72]. Once we have different algorithm formulations for the RTM, we wrap them into the Monte Carlo (MC)

---

<sup>1</sup>We use the word cell in this context to refer to the grid point or element of the discretized wave equation using, for instance, the Finite Difference or Finite Element Methods, respectively.

sampling method. In this scenario, it is necessary a set of velocity fields (samples) that has a direct relation to the seismic image outcomes after migration. Iterating the RTM over the number of samples is even more challenging because the computational cost grows with the number of iterations [10, 47].

As mentioned earlier, RTM algorithms take advantage of heterogeneous machines based on emerging multi-CPU, multi-GPU, and multi-VPs technologies. We evaluate the computational performance of different RTM algorithms in terms of execution time, storage demand, overhead, and speedup. Analyses based on profiling tools such as the `ftrace` and `nvprof` show that our computational implementations have high rates of vectorization and GPU usage as well as extremely low cache missing ratios. Furthermore, even in situations that provide high overheads due to the time spent on compressing the source wavefield, if we choose an adequate platform it is possible to reduce execution time and storage demand. This characteristic is strongly present in the RTM with wavefield reconstruction. Such algorithm strategy provides the best execution time and storage demand for all platforms. Although the RTM with wavefield reconstruction methodology requires an extra wave equation to be solved the effects show that its computational performance is the best one without hampering the final seismic images. The same performance analysis can be done for the RTM wrapped into the MC method. Thus, we compare different RTM algorithmic developments aiming to show the benefits of the wavefield reconstruction for the RTM on co-processors like GPUs and VPs instead of the classical CPUs.

## 1.2 Outline

Based on what we expose in the introduction and objective sections, this work is laid out as follows. Chapter 2 introduces basic concepts about seismic imaging and the mathematical description concerning the generalized FWI. After that, we derive the RTM technique by simplifying the generalized FWI equations. RTM is the one used in the work to present different algorithmic approaches. Chapter 3 presents how we can decrease the computational time for solving the wave equation by applying a high-order finite difference discretization to the governing PDEs. Besides, it describes two strategies to reduce wavefield storage, one based on wavefield compression and another one based on wavefield reconstruction methods. Different from Chapter 3, Chapter 4 presents the computational implementations for different platforms, such as multi-CPU, multi-VP, and multi-GPU machines. In this case, it is described the C language implementations for the key parts of the RTM algorithms in three different computational environments. Chapter 5 presents the computational performance analysis in terms of execution time, speedup, and storage demand for the seismic modeling and RTM on three different computational platforms, the RTM technique which applies data compression to seismograms and wavefields, and for a wrapped RTM into the MC sampling method that can be used for

uncertainty quantification on seismic images. Finally, Chapter 6 presents the conclusions and possible future research directions.

### 1.3 Contributions and Published Materials

This study is based on the following materials that are either published after peer review, in pre-print, or in preparation with joint authorship:

- Carlos H.S. Barbosa, Liliane N.O. Kunstmann, Rômulo M. Silva, Charlan D.S. Alves, Bruno S. Silva, Djalma M.S. Filho, Marta Mattoso, Fernando A. Rochinha, Alvaro L.G.A. Coutinho, "A workflow for seismic imaging with quantified uncertainty", *Computers & Geosciences*, v. 145, pp. 104615, 2020.
- Carlos H.S. Barbosa, Alvaro L.G.A. Coutinho, "Enhancing Reverse Time Migration: Hybrid Parallelism plus Data Compression". In: *Proceedings of the XLI Ibero-Latin-American Congress on Computational Methods in Engineering*. ABMEC, November, 2020.
- Rodolfo S.M. Freitas, Carlos H.S. Barbosa, Gabriel M. Guerra, Alvaro L.G.A. Coutinho, Fernando A. Rochinha, "An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty", *Computational Geosciences*, v. 25, n. 3, pp. 1229-1250, 2021.
- Carlos H.S. Barbosa, Alvaro L.G.A. Coutinho, "Multi-GPU 3-D Reverse Time Migration with Minimum I/O", *High Performance Computing*, pp. 160-173, Cham, September. Springer International Publishing, 2022. Best Paper Award - CARLA2022.
- Carlos H.S. Barbosa, Alvaro L.G.A. Coutinho, "Seismic Modeling and Migration with Random Boundaries on the NEC SX-Aurora TSUBASA". In Preparation.

# Chapter 2

## Seismic Imaging

Seismic imaging is one of the most effective ways to map subsurface geological structures and physical properties using seismic data. The practice of exploration seismology in the Oil&Gas industry has basically established three approaches for seismic imaging: pre-processing to improve the data signal-to-noise ratio as well as retain the primary reflection energy from the full wavefield data, building a long-wavelength reference velocity parameter model using velocity model building (VMB) techniques, and using seismic migration methods to map the short-wavelength reflectivity structures. Thus, the reflectivity mapping that builds a representation of the Earth's subsurface consists of repositioning the recorded data into its "true" geological position in the subsurface. The representation of the Earth's subsurface is referred to seismic image (or only image). Besides, we highlight the word true because the "true" geological position strongly depends on the Earth parameters used in the seismic migration. Among them, we can cite the velocity, dip, and anisotropy distributions [30, 59, 60].

There are two main categories of migration algorithms, which are the integral methods and the differential methods. Integral techniques, such as Kirchhoff, equivalent-offset, common reflection angle, and beam migrations, solve a high-frequency approximation of the wave equation. The solution is treated as a spike-like event, where the summation of these events, with appropriate amplitude scaling, reconstructs the final seismic image by superposition of the stationary phase components [14]. Integral methods take into account a simplified description of the wave propagation called "rays". The "rays" are vectors that indicate the direction and arrival times of the wavefront motion along the associated ray-path [59]. On the other hand, the differential methods use wavefield extrapolation to solve the wave equation. In this case, the wave motion is described by differential equations and their solution is referred to wavefield extrapolation [59]. Differential methods include the wavefield extrapolation migration (WEM), also referred to "wave equation migration", the Phase Shift, Phase Shift plus interpolation, and Reverse Time Migration (RTM). Both classes of migration can be performed either in time or depth domain [14]. The main difference between them is that time migration techniques assume the velocity distribu-

tion to be laterally invariant whilst depth migrations honor the lateral velocity changes to at least first order. Another difference between time and depth migrations is that time migration outputs the image in two-way travel time. Conversely, depth migration outputs its image in apparent vertical depth if all relevant Earth parameters are used correctly in the migration [59, 60].

Despite its name, RTM is a depth migration method based on the two-way wave equation and an imaging condition that provides a reliable high-resolution representation of the Earth’s subsurface [30, 103, 114]. RTM and most other migration methods are one-step methods aiming at imaging geological structures rather than the true amplitude of the Earth’s reflectivity [103, 135]. To overcome this limitation, RTM can be imaging the subsurface via an iterative inversion scheme. The inversion process leads to two different formulations called Least-Squares Migration (LSM) and Full Waveform Inversion (FWI). In the first one the smooth background parameter model is the same at every iteration. Then the final reflectivity image is known as the least-squares migration image. On the other hand, both the high-wavenumber reflectivity and the smooth parameter model are updated after each iteration then the convergence yields the FWI tomogram, also known as the nonlinear inversion image [103]. If we take a step further to modify the FWI to output the reflectivity, we refer to this approach as FWI Imaging and the resulting reflectivity parameter model as FWI Image [34, 57, 120, 130].

In this chapter, we describe the generalized FWI, which provides the full description of seismic imaging to estimate the model parameter and reflectivity distributions of the Earth from recorded seismic data. After that, we present the RTM technique from the generalized FWI, where its first iteration produces the RTM image if the background parameter model does not generate reflection [33, 68, 69, 103, 115, 135]. The main equations which describe the RTM are the focus of the present work.

## 2.1 Generalized Full Waveform Inversion

FWI is defined as a mathematical optimization problem that aims to minimize a misfit function  $E$ , which measures the difference between seismic data acquired in a seismic survey and the seismic data numerically simulated by wave equations. Using the  $\ell^2$ -norm as the metric error, the misfit function for the measured and simulated seismic data predicted from a 3D parameter model  $\mathbf{m}$  is written as [114]:

$$E(\mathbf{m}) = \frac{1}{2} \int_0^T [p(\mathbf{r}_r, t) - s(\mathbf{r}_r, t)]^2 dt, \quad (2.1)$$

where  $\mathbf{r}$  are the spatial coordinates,  $t$  the time in  $[0, T]$ ,  $s(\mathbf{r}_r, t)$  and  $p(\mathbf{r}_r, t)$  represent the measured and simulated seismic signals recorded at the receiver positions  $\mathbf{r}_r$ . Both measured and simulated seismic signals induced by a seismic source (or shot) are called

seismograms, and we will distinguish them as measured and simulated seismograms.

As mentioned earlier, the simulated seismogram  $p(\mathbf{r}_r, t)$  can be predicted by solving a wave equation. Restricting ourselves to the acoustic wave equation,  $p(\mathbf{r}, t)$  is determined by:

$$\nabla^2 p(\mathbf{r}, t) - \sigma(\mathbf{r})^2 \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} = f(t) \delta(\mathbf{r} - \mathbf{r}_s), \quad (2.2)$$

where  $p(\mathbf{r}, t)$  is the pressure,  $\sigma(\mathbf{r}) = 1/v(\mathbf{r})$  is the slowness with  $v(\mathbf{r})$  the velocity for the compressional wave,  $f(t)$  is the seismic source at the position  $\mathbf{r} = \mathbf{r}_s$ , and  $\delta$  is the Dirac delta function. The pressure  $p$ , and the slowness  $\sigma$  are defined in a domain  $\Omega \subset \mathbb{R}^{n_{sd}}$ ,  $n_{sd} = 2, 3$ . Note that the parameter model for the acoustic wave equation contains only the slowness, that is,  $\mathbf{m} = \{\sigma\}$ . Although Equation 2.2 provides the solution in  $\Omega$ , for seismic imaging applications the misfit function is computed only at the receiver location  $\mathbf{r}_r$ , which is  $p(\mathbf{r}_r, t)$ . Notice that the parameter model for Equation 2.2 is the slowness field. The second-order differential equation (2.2) needs initial and boundary conditions. A natural initial condition is to define  $p(\mathbf{r}, 0) = \partial p(\mathbf{r}, 0) / \partial t = 0$  for  $\mathbf{r} \in \Omega$ . Thus, we set  $p(\mathbf{r}, t) = 0$  on  $\partial\Omega$ , the domain boundary.

Solving the wave equation in finite domains generates artificial reflections coming from the boundaries. Normally, it is common to use additional equations to eliminate such artificial events. Among the several options in the literature, the Convolutional Perfectly Matched Layer (CPML) [63, 97] and the damping factors for plane waves introduced by CERJAN *et al.* [25] are the most used. Although unusual in wave propagation simulation studies, the RBCs, first introduced by CLAPP [32], can also be employed in seismic imaging methods based on the two-way wave equation, such as RTM and FWI [31, 32, 72, 93]. We detail the three main boundary conditions used in the literature in Appendix A.

Minimization of the misfit function (Equation 2.1) is achieved iteratively due to its nonlinear behavior. Thus, the model update can be obtained by:

$$\sigma_{k+1}(\mathbf{r}) = \sigma_k(\mathbf{r}) - \alpha_k \rho_k(\mathbf{r}), \quad (2.3)$$

where  $k$  is the iteration counter,  $\alpha_k$  is known as the step size computed by line search algorithms [94], and  $\rho_k(\mathbf{r})$  is a search direction, determined by gradient methods, such as the conjugated gradient, quasi-Newton or Newton methods [94]. Using the adjoint-state method [68, 114], the gradient for the misfit function in Equation 2.1 can be written as the convolution between the calculated (its second-time derivative) and adjoint state wavefields [99, 103], as follows:

$$g(\mathbf{r}) = 2\sigma(\mathbf{r}) \int_0^T \bar{p}(\mathbf{r}, \tau) \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} dt, \quad (2.4)$$

where  $\tau = T - t$  is the reversal time evolution defined as in GIVOLI [52], with  $\tau \in [0, T]$ ,

and the adjoint-state field  $\bar{p}(\mathbf{r}, \tau)$  is found by solving the following PDE:

$$\nabla^2 \bar{p}(\mathbf{r}, \tau) - \sigma(\mathbf{r})^2 \frac{\partial^2 \bar{p}(\mathbf{r}, \tau)}{\partial \tau^2} = \Delta p(\mathbf{r}_r, \tau) \delta(\mathbf{r} - \mathbf{r}_r), \quad (2.5)$$

where  $\Delta p(\mathbf{r}_r, \tau) = s(\mathbf{r}_r, \tau) - p(\mathbf{r}_r, \tau)$  is the difference between the measured and simulated wavefields at the receiver positions  $\mathbf{r} = \mathbf{r}_r$ . Notice that  $\bar{p}$  is also defined in  $\Omega \subset \mathbb{R}^{n_{sd}}$ , and the corresponding initial condition is defined as  $\bar{p}(\mathbf{r}, 0) = \partial \bar{p}(\mathbf{r}, 0) / \partial \tau = 0$  for  $\mathbf{r} \in \Omega$ . Lastly,  $\bar{p}(\mathbf{r}, \tau) = 0$  on  $\partial\Omega$ . Recall that, we must use ABCs in equation 2.5 to avoid artificial reflections on the boundaries. Any of the options detailed in Appendix A can be used with equation 2.5.

Considering the Newton method, the full description for  $\rho_k(\mathbf{r})$  takes into account the Hessian matrix  $\mathbf{H}_k$ , which consists of second partial derivatives of  $E(\mathbf{v})$ . However, the most accurate inversion algorithms, such as the Newton method, demand too much computational resources. As an alternative, it is common to use in academia and industry the quasi-Newton and L-BFGS methods [78, 88]. Here we choose to use the gradient of  $E(\mathbf{v})$  preconditioned by the diagonal of the pseudo Hessian for  $\rho_k(\mathbf{r})$  as proposed by CARNEIRO *et al.* [23]. Take into account the reversal time evolution [52], we re-write the gradient preconditioned by the diagonal of the pseudo Hessian as follows:

$$\tilde{\mathbf{H}}_k^{-1}(\mathbf{r}) g_k(\mathbf{r}) = -\frac{2}{\sigma(\mathbf{r})} \frac{\int_0^T \bar{p}(\mathbf{r}, \tau) \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} dt}{\int_0^T \left| \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \right|^2 dt}. \quad (2.6)$$

According to CARNEIRO *et al.* [23] the preconditioning of Equation 2.6 is particularly important for increasing the amplitudes on the deepest targets due to the self correlation term.

### 2.1.1 Reverse Time Migration

Following SCHUSTER [103], the migration image can be defined as the residual slowness:

$$\begin{aligned} I_S(\mathbf{r}) = \sigma_1(\mathbf{r}) - \sigma_0(\mathbf{r}) &= -\alpha_0 \rho_0(\mathbf{r}), \\ &= \frac{2}{\sigma(\mathbf{r})} \frac{\int_0^T \bar{p}(\mathbf{r}, \tau) \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} dt}{\int_0^T \left| \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \right|^2 dt}, \end{aligned} \quad (2.7)$$

if the background slowness model  $\sigma_0(\mathbf{r})$  does not generate reflections. On the other hand, CLAERBOUT [30] suggests that the migration image can be expressed as:

$$I_C(\mathbf{r}) = \frac{\int_0^T \bar{p}(\mathbf{r}, \tau) p(\mathbf{r}, t) dt}{\int_0^T |p(\mathbf{r}, t)|^2 dt}. \quad (2.8)$$

The imaging conditions  $I_S(\mathbf{r})$  and  $I_C(\mathbf{r})$  approximate the Earth's reflectivity distribution. Further, if we solve Equations 2.2 and 2.5 with numerical methods, such as the Finite Difference Method,  $I_S(\mathbf{r})$  and  $I_C(\mathbf{r})$  are known as the Reverse Time Migration image [103]. The relation between the imaging conditions from equations 2.7 and 2.8 will be explored in the next chapter.

Note that the migration image in equation 2.7 corresponds to the first iteration of the generalized FWI solved by the iterative process defined in equation 2.3. Besides, if we fix the background parameter model over the iterations, as discussed earlier, the iterative RTM formulation leads to the LS-RTM formulation, and equation 2.7 is its first iteration [103]. Therefore, we expect that advances in the computational implementation of the RTM can be equally applied to the LS-RTM and FWI technologies.

# Chapter 3

## Computational Strategies to Boost Reverse Time Migration

According to ZHOU *et al.* [135], the RTM's main challenges are related to the application of imaging conditions, imaging artifacts suppression, velocity model building based on migration, and computational efficiency improvements. Improving computational efficiency becomes one of the key challenges for applying 3-D high resolution RTM in the industry. The wave equation solution in seismic imaging methods, such as RTM, dominates the computational cost in terms of time and storage. In this sense, we present in this chapter how we can decrease the computational time for solving the wave equation by applying a high-order finite difference discretization to the governing Partial Differential Equations (PDEs). Besides, we describe two strategies to reduce wavefield storage. We implement the wavefield compression and its reconstruction by adding a new equation to the problem. The high-order discretization for the wave equation, wavefield compression, and wavefield reconstruction are presented in sections 3.1, 3.2, and 3.3.

### 3.1 High-order finite difference method for wavefield propagation

The numerical discretization used to solve the acoustic wave equation (2.2) is based on the non-staggered grid scheme (NSG) [39]. The NSG is the most simple and popular finite-difference numerical scheme used to discretize Partial Differential Equations (PDEs) where each derivative can be approximated using numerical operators obtained by Taylor series expansions [39]. However, the numerical implementation based on the Finite Difference Method (FDM) suffers from numerical stability and numerical dispersion, which can be alleviated by simultaneously squeezing the grid size and time interval. On the other hand, the computational cost of a single solution of the wave equation is strongly related to the number of grid points for which the wavefield must be computed.

There are many strategies to reduce the impact of solving the wave equation in imaging techniques to simulate a seismic acquisition with hundreds of thousands of shots. Among them, the information redundancy can be reduced by gathering the shots (shot gathers with simultaneous shots) or by migrating plane-waves [80, 107]. Another way is truncating the domain around the acquisition configuration or the wavefront domain, expanding it as the wavefield propagates [95, 96].

In this section, we aim to reduce the computational cost to solve the wave equation by applying a high-order finite difference discretization to the wave equation. The high-order finite difference discretization allows for reducing the number of grid points for which the wavefield must be computed. Although the number of arithmetic operations increases per point, the total time required for solving the wave equation decreases. We would like to mention that there are other ways to increase the performance efficiency of the wave equation FD discretization implementation by, for instance, loop transformations such as permutation, fission, collapsing, space, and temporal tiling among others [1, 48, 58, 98, 100, 104, 111]. The idea involves the modification of the data flow to improve the reuse of loaded data on the hierarchical memory (registers, cache, and RAMs) and parallelism [122, 123]. Although the high-order discretization and loop transformation techniques can be implemented together, we focus initially on the first one to exemplify the relation between execution time and memory usage for different wave field discretization orders.

We use the standard approach Taylor series to reach the high-order finite difference approximations. Thus, let  $\phi(x, y, z, t)$  be a function in  $\mathbb{R}^4$  with  $a \leq x \leq b$ ,  $c \leq y \leq d$ ,  $e \leq z \leq f$ , and  $g \leq t \leq h$ . The intervals  $[a, b]$ ,  $[c, d]$ ,  $[e, f]$ , and  $[g, h]$  are sets following  $a = x_0, \dots, x_i, \dots, x_{N+1} = b$ ,  $c = y_0, \dots, y_j, \dots, y_{N+1} = d$ ,  $e = z_0, \dots, z_k, \dots, z_{N+1} = f$ , and  $g = t_0, \dots, t_n, \dots, t_{N+1} = h$ . The discrete version for the function  $\phi(x, y, z, t)$  is  $\phi(x_i, y_j, z_k, t_n) = [\phi(a, c, e, g), \phi(x_1, y_1, z_1, t_1), \dots, \phi(x_i, y_j, z_k, t_n), \dots, \phi(b, d, f, h)]$ , where the value of  $\phi(x_i, y_j, z_k, t_n)$  can be represented by  $\phi_{i,j,k}^n$ . The values of  $\Delta x = x_{i+1} - x_i$ ,  $\Delta y = y_{j+1} - y_j$ , and  $\Delta z = z_{k+1} - z_k$  represent the grid spaces, and  $\Delta t = t_{n+1} - t_n$  represents the temporal sampling rate. For a uniform grid  $\Delta x = \Delta y = \Delta z = \Delta t = (b - a)/(N + 1) = (d - c)/(N + 1) = (f - e)/(N + 1) = (h - g)/(N + 1)$ . Every grid point  $(x_i, y_j, z_k, t_n)$  is represented in the grid by the indexes  $(i, j, k, n)$  and the neighbors relative to these grid points represented by the indexes  $(i \pm 1, j \pm 1, k \pm 1, n \pm 1)$ ,  $(i \pm 2, j \pm 2, k \pm 2, n \pm 2)$ ,  $\dots$ . Expanding each of the function values of  $\phi(x_i, y_j, z_k, t_n)$  in a Taylor series about the points  $x_i$ ,  $y_j$ ,  $z_k$ , and  $t_n$  gives:

$$\phi(x_i, y_j, z_k, t_n + \Delta t) = \sum_{m=0}^{\infty} \frac{(\Delta t)^m}{m!} \frac{\partial^m \phi(x_i, y_j, z_k, t_n)}{\partial t^m} + O((\Delta t)^{m+1}), \quad (3.1)$$

$$\phi(x_i + \Delta x, y_j, z_k, t_n) = \sum_{m=0}^{\infty} \frac{(\Delta x)^m}{m!} \frac{\partial^m \phi(x_i, y_j, z_k, t_n)}{\partial x^m} + O((\Delta x)^{m+1}), \quad (3.2)$$

$$\phi(x_i, y_j + \Delta y, z_k, t_n) = \sum_{m=0}^{\infty} \frac{(\Delta y)^m}{m!} \frac{\partial^m \phi(x_i, y_j, z_k, t_n)}{\partial y^m} + O((\Delta y)^{m+1}), \quad (3.3)$$

$$\phi(x_i, y_j, z_k + \Delta z, t_n) = \sum_{m=0}^{\infty} \frac{(\Delta z)^m}{m!} \frac{\partial^m \phi(x_i, y_j, z_k, t_n)}{\partial z^m} + O((\Delta z)^{m+1}). \quad (3.4)$$

Using the definition of equations 3.1, 3.2, 3.3, 3.4, expanding the function  $\phi(x_i, y_j, z_k, t_n)$  around the grid points  $x_i, y_j, z_k$ , and  $t_n$  for the neighbors relative to the grid point  $(i, j, k, n)$  such as  $(i \pm 1, j \pm 1, k \pm 1, n \pm 1)$ ,  $(i \pm 2, j \pm 2, k \pm 2, n \pm 2)$ ,  $\dots$ , and combining the resulting equations, lead to the generalized second-order discretizations for the function  $\phi(x, y, z, t)$ :

$$\frac{\partial^2 \phi(x, y, z, t)}{\partial t^2} \approx \frac{1}{\Delta t^2} \left[ b_0 \phi_{i,j,k}^n + \sum_{m=1}^{N/2} b_m (\phi_{i,j,k}^{n+m} + \phi_{i,j,k}^{n-m}) \right] + O((\Delta t)^{m+2}), \quad (3.5)$$

$$\frac{\partial^2 \phi(x, y, z, t)}{\partial x^2} \approx \frac{1}{\Delta x^2} \left[ b_0 \phi_{i,j,k}^n + \sum_{m=1}^{N/2} b_m (\phi_{i+m,j,k}^n + \phi_{i-m,j,k}^n) \right] + O((\Delta x)^{m+2}), \quad (3.6)$$

$$\frac{\partial^2 \phi(x, y, z, t)}{\partial y^2} \approx \frac{1}{\Delta y^2} \left[ b_0 \phi_{i,j,k}^n + \sum_{m=1}^{N/2} b_m (\phi_{i,j+m,k}^n + \phi_{i,j-m,k}^n) \right] + O((\Delta y)^{m+2}), \quad (3.7)$$

$$\frac{\partial^2 \phi(x, y, z, t)}{\partial z^2} \approx \frac{1}{\Delta z^2} \left[ b_0 \phi_{i,j,k}^n + \sum_{m=1}^{N/2} b_m (\phi_{i,j,k+m}^n + \phi_{i,j,k-m}^n) \right] + O((\Delta z)^{m+2}). \quad (3.8)$$

The values of  $b_m$  are related to the finite difference order discretization, and  $N$  is an integer greater or equal to two. Table 3.1 shows the finite-difference coefficients for the second derivative discretization. Besides, each order discretization has its finite-difference coefficients. All the coefficient values can be obtained by the Taylor expansion based on equations 3.1, 3.2, 3.3, and 3.4.

Table 3.1: Finite-difference coefficients for the second derivative discretization.

N	$b_0$	$b_1$	$b_2$	$b_3$	$b_4$	$b_5$
2	-2	1				
4	$-\frac{5}{2}$	$\frac{4}{3}$	$-\frac{1}{12}$			
6	$-\frac{49}{18}$	$\frac{3}{2}$	$-\frac{3}{20}$	$\frac{1}{90}$		
8	$-\frac{205}{72}$	$\frac{8}{5}$	$-\frac{1}{5}$	$\frac{8}{315}$	$-\frac{1}{560}$	
10	$-\frac{5269}{1800}$	$\frac{5}{3}$	$-\frac{5}{21}$	$\frac{5}{126}$	$-\frac{5}{1008}$	$\frac{5}{3150}$

Applying the finite difference scheme to the equation 2.2 that is 2nd-order in space and 2nd-order in time produces the following discretization:

$$\begin{aligned}
p_{i,j,k}^{n+1} = C_{i,j,k} [2b_0 + & \tag{3.9} \\
& \sum_{m=1}^1 b_m (p_{i+m,j,k}^n + p_{i-m,j,k}^n + p_{i,j+m,k}^n \\
& + p_{i,j-m,k}^n + p_{i,j,k+m}^n + p_{i,j,k-m}^n) \\
& - 2p_{i,j,k}^n + p_{i,j,k}^{n-1} - f_{i,j,k}^n,
\end{aligned}$$

where the indexes  $i$ ,  $j$  and  $k$  represent the directions  $(x, y, z)$ ,  $n$  is the temporal step, and  $p$  is the pressure. The matrix  $C_{i,j,k} = (v_{i,j,k} \Delta t / h)^2$ , where  $v_{i,j,k}$  is the velocity, and  $h = \Delta x = \Delta y = \Delta z$  is the grid size. Lastly,  $b_m$  are the finite difference coefficients presented in Table 3.1 for  $N = 2$ .

Expressions 3.5, 3.6, 3.7, and 3.8 are the most general discretization form for the partial derivatives of the function  $\phi(x, y, z, t)$ . Because of that, if we aim to implement the wave equation for different FD orders, then we need to change only the values of  $N$ . The 4th-order and 8th-order in space with 2nd-order in time for equation 2.2 are given by the following expression:

$$\begin{aligned}
p_{i,j,k}^{n+1} = C_{i,j,k} [2b_0 + & \tag{3.10} \\
& \sum_{m=1}^2 b_m (p_{i+m,j,k}^n + p_{i-m,j,k}^n + p_{i,j+m,k}^n \\
& + p_{i,j-m,k}^n + p_{i,j,k+m}^n + p_{i,j,k-m}^n) \\
& - 2p_{i,j,k}^n + p_{i,j,k}^{n-1} - f_{i,j,k}^n,
\end{aligned}$$

$$\begin{aligned}
p_{i,j,k}^{n+1} = C_{i,j,k} [2b_0 + & \tag{3.11} \\
& \sum_{m=1}^4 b_m (p_{i+m,j,k}^n + p_{i-m,j,k}^n + p_{i,j+m,k}^n \\
& + p_{i,j-m,k}^n + p_{i,j,k+m}^n + p_{i,j,k-m}^n) \\
& - 2p_{i,j,k}^n + p_{i,j,k}^{n-1} - f_{i,j,k}^n.
\end{aligned}$$

Before discussing the relation between the FD order discretization and its computational cost, we will present in the next section the numerical implementation of the acoustic wave equation, which is the RTM kernel.

### 3.1.1 Numerical implementation of acoustic wave equation

Several tools in the scientific community provide high-level abstraction layers to lead with efficient computational implementations of scientific simulations. Among them, the Devito [84], FEniCS [81], Snowflake [128], SBLI [91], Saiph [85], and others Domain Specific Languages (DSLs) allow simulating complex physical systems based on PDEs. The scientific applications are vast, and it is common to see developments in computational fluid dynamics [81, 128], seismic inversion problems [81, 84], heat flow applications [81, 128], shock-wave/boundary-layer interaction problems [91] among others. Specifically related to our work, the Devito DSL is a code generation tool that provides numerical discretizations based on the FDM for PDEs of seismic inversion problems. Devito uses symbolic Python to express complex mathematical operators and exploits modern HPC architectures by automating performance and domain-specific optimizations providing optimized parallel codes based on SIMD vectorization, CPU and GPU parallelism via OpenMP and OpenACC, multi-node parallelism via MPI, blocking, etc. [84]. Although Devito DSL provides different approaches to optimize the computational codes, we judge extremely important to develop ours aiming to provide specific advances for current computational architectures (based on CPUs and GPUs, for example) and mainly for the emerging ones like vector processors. Therefore, the following paragraphs and sections describe the algorithmic strategies for wave propagation and RTM problems based on the acoustic wave equation in an isotropic medium.

The FDM discretization of the acoustic wave equation leads to the discrete versions of the velocity field, forward-propagated wavefield (source wavefield), and seismic source represented by the vectors  $\mathbf{v}$ ,  $\mathbf{p}$ , and  $\mathbf{f}$ , respectively. For the 3D case, the vectors  $\mathbf{v}$  and  $\mathbf{p}$  have the dimension  $N = N_x \times N_y \times N_z$ , where  $N_x$ ,  $N_y$  and  $N_z$  are the number of grid points in each Cartesian direction. Lastly, the seismic source  $\mathbf{f}$  has dimension  $N_t$  for each shot, where  $N_t = T/\Delta t$ .

We call the process of numerically solving the wave equation seismic modeling, which consists of simulating the wavefield propagation for a given domain. Thus, the description of the seismic modeling algorithm is quite simple once we have the discretized version of the wave equation. Algorithm 1 shows that for the acoustic wave equation, only two inputs are needed. The first one is a velocity field  $\mathbf{v}$  representing the spatial velocity distribution for the compressional wave (P-wave), and the second is a vector  $\mathbf{f}$  containing information about the seismic signature, called seismic source, responsible for initiating the wave propagation through the medium. The wave equation propagation is solved over a temporal loop (the inner loop of Algorithm 1) for each *shot\_id* (loop in line 3). The shot refers to the seismic source that starts the wave propagation, and each one is localized in the domain represented by the finite-difference grid. A computational implementation of ABCs leads to non-spurious reflections on truncated domains. We detail the numeri-

cal implementation of boundary conditions for truncated domains in Appendix A. Note that the wave equation solutions for each shot are independent. Therefore they can be categorized as an embarrassingly parallel problem [41].

---

**Algorithm 1** Seismic modeling for the wavefield propagation

---

**Require:**  $\mathbf{v}$ , and  $\mathbf{f}$

```

1: function SEISMIC_MODELING( vector  $\mathbf{v}$ , vector  $\mathbf{f}$  )
2:   read  $\mathbf{v}$ , and  $\mathbf{f}$ 
3:   for  $shot\_id = 1$  to  $N_{shots}$  do
4:     initialize  $n_t = 0$ 
5:     apply initial conditions for  $i_t = 0$ 
6:     for  $i_t = 1$  to  $N_t$  do
7:        $n_t = i_t * \Delta t$ 
8:       solve equation (2.2)
9:       apply absorbing boundary condition
10:      store  $\mathbf{p}_{n_t}$  for all  $n_t$ 
11:    end for
12:    store  $\mathbf{s}_{shot\_id}$ 
13:  end for
14: end function

```

---

To illustrate the use of Algorithm 1, Figure 3.1 presents an example of the wavefield propagation in a constant 3D velocity field of 3000.0 m/s. The three time-frames refer to the wavefield propagation instants 0.6 s (upper), 1.0 s (middle), and 1.5 s (lower) for a single shot located at  $[x, y, z] = [5000.0, 5000.0, 25.0]$  meters. We use the Ricker wavelet with a cutoff frequency of 20.0 Hz as the seismic source. We did not use any ABC in this numerical test because we aim to only exemplify the wave propagation in a domain with constant velocities. To avoid artificial reflections on the boundaries, we limit the total propagation time of the seismic wavefield to 1.7s.

### 3.1.2 Computational cost of high-order discretizations

The FD parameters such as time sampling and grid spacing affect the computational costs and storage requirements significantly. It is, therefore, worthwhile to determine their maximum values to guarantee the best performance of seismic imaging applications. However, there are limits for the time sampling and grid space related to the Partial Differential Equations (PDEs) that govern the wavefield propagation phenomenon due to numerical dispersion, and stability [92]. For instance, the largest value of the time sampling interval, which can be used in a given model without making the system numerically unstable is given by:

$$\Delta t = \frac{\mu \max(\Delta x, \Delta y, \Delta z)}{v_{max}}, \quad (3.12)$$

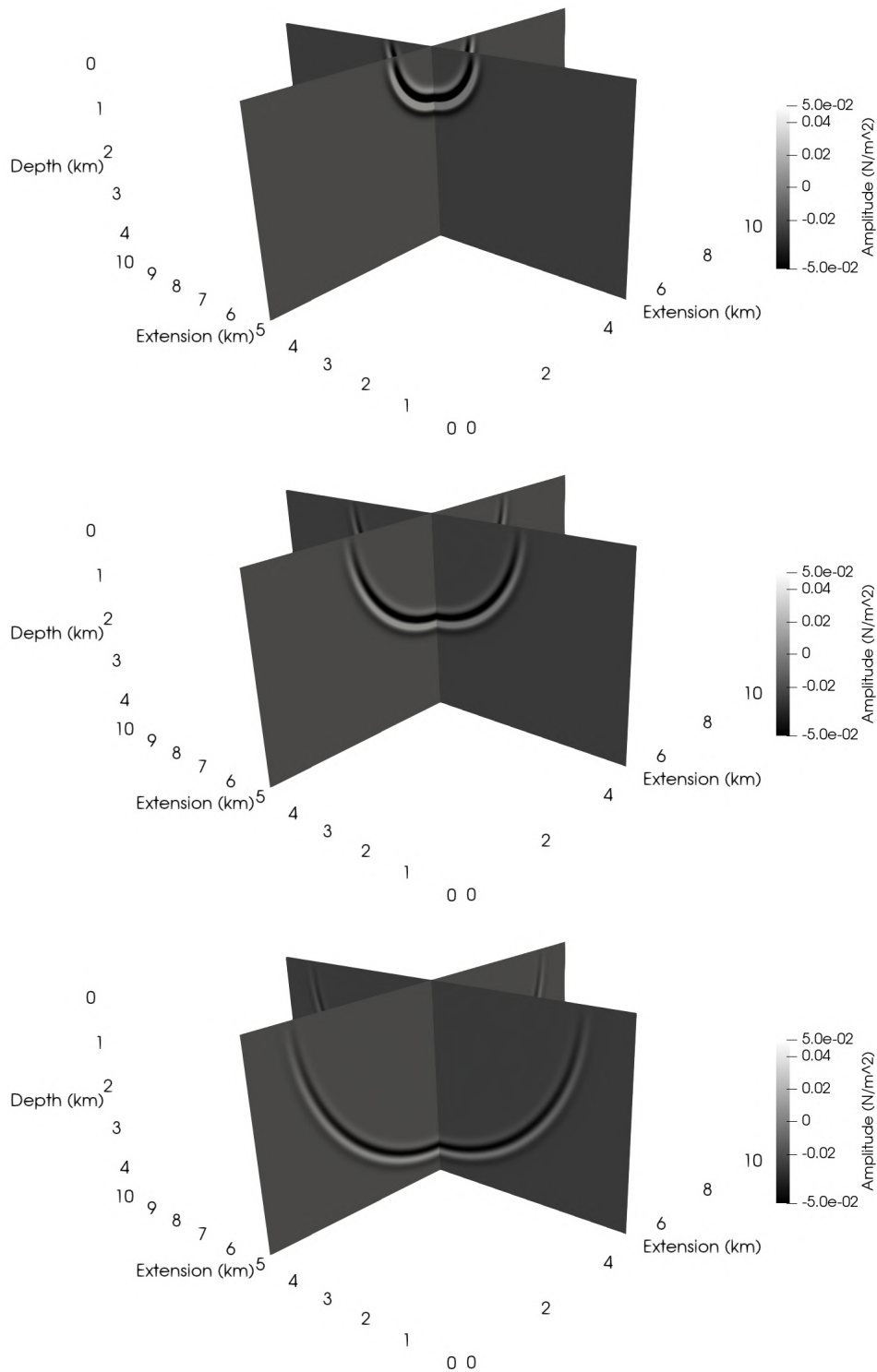


Figure 3.1: Propagation of the wavefield in the constant 3D velocity field of 3000.0 m/s for the instants 0.6 s (upper), 1.0 s (middle), and 1.5 s (lower).

where  $v_{max}$  is the highest velocity value of the medium, and  $\mu$  is a parameter that depends on the algorithm used for computing the wavefield.

On the other hand, the maximum value of grid spacing which can be used in a model without causing excessive dispersion of energy is governed by the relation:

$$\max(\Delta x, \Delta y, \Delta z) = \frac{v_{min}}{G f_{max}}, \quad (3.13)$$

where  $v_{min}$  is the lower velocity value of the medium,  $f_{max}$  the maximum cut-off frequency in the seismic source, and  $G$  the number of samples per wavelength corresponding to  $f_{max}$ .

As mentioned earlier, the value of  $\mu$  depends on the numerical method used to discretize the wave equation. A common way to determine  $\mu$  is by following the von Neumann analysis [70, 92]. For an FD scheme that is 2nd-order accurate in time and 4th-order in space,  $\mu$  assumes the value of 0.5. The  $G$  factor, which controls the numerical dispersion also depends on the numerical method used to discretize the wave equation. Thus, using the same FD scheme that is 2nd-order accurate in time and 4th-order in space,  $G = 5$ .

Many works study how to determine the values of  $\mu$ ,  $G$ , and the FD coefficients depending on the order of discretization for wave equations [22, 27–29, 92, 108]. Because of that, we aim to show in this section, in the practical point view, that increasing the FD order of discretization of the wave equation reduces the spatial grid density. Thus, our test case consists of running the seismic modeling based on Algorithm 1 for two different grid sizes and three different discretization FD orders. The seismic source with a cutoff frequency of 30.0Hz is located at [5.0, 5.0, 0.025] km, and a single receiver located at [5.0, 5.0, 2.95] km records one seismic signal for each grid size/FD-order configuration. We use a parameter model with a constant velocity of 3000.0 m/s.

Figures 3.2, 3.3, and 3.4 show the seismic signal recorded at [5.0, 5.0, 2.95] km for the acoustic wave equation that is 2nd-order accurate in time, and 2nd-order (Figure 3.2), 4th-order (Figure 3.3), and 8th-order (Figure 3.4) accurate in space. Besides, we solve the wave equation for two grid sizes with different grid spaces, one has  $401 \times 401 \times 181$  points for the grid space of 25.0 meters, and the other  $801 \times 801 \times 361$  points for the grid space of 12.5 meters. Figures 3.2 and 3.3 lower show that the recorded seismic signals for the 2nd-order accurate in space (grid spaces of 12.5 and 25.0 meters) and 4th-order accurate in space (grid space of 25.0 meters) present numerical dispersion. On the other hand, Figures 3.3 upper and 3.4 show that the recorded seismic signals for the 4th-order accurate in space (grid space of 12.5 meters) and 4th-order accurate in space (grid spaces of 12.5 and 25.0 meters) are accurate enough to do not present numerical dispersion. Considering the results without dispersion, we could implement an acoustic wave equation with the 4th-order accurate in space with a grid space of 12.5 meters or the 8th-order accurate in space with a grid space of 25.0 meters. However, the wave equation with the 4th-

order accurate in space demands eight grid points more than the wave equation with the 8th-order accurate in space. Thus, although the 8th-order discretization requires more arithmetic operations to update the space-time points than the 4th-order discretization, the total number of grid points to be updated is lower.

Table 3.2 shows the execution time of the seismic modeling<sup>1</sup>, where the wave equation discretization is 2nd-order accurate in time, and 2nd-order, 4th-order, and 8th-order accurate in space for the grids  $401 \times 401 \times 181$  and  $801 \times 801 \times 361$ . The execution times in blue indicate the wave equation solution without numerical dispersion. Following the measured execution times in blue and the numerical dispersion analysis, the results suggest that the 2nd-order accurate in time and 8th-order accurate in space implementation is the best choice for the wave equation discretization in terms of computational efficiency and wavefield solution.

Table 3.2: Execution time of seismic modeling for three FD-order discretizations on two different grid sizes. The blue execution times indicate the wave equation solution without numerical dispersion.

Grid / FD-order	2nd	4th	8th
$401 \times 401 \times 181$	23.119s	27.871s	36.615s
$801 \times 801 \times 361$	191.298s	233.866s	371.231s

Theoretically, it is possible to increase the FD discretization order and, consequently, the grid space, but the computational grid can not represent smaller geological structures. So, it is essential to understand the problem to avoid missing important information due to large grid spaces, and after choose the best discretization order that gives its best performance for the proposed scenario.

<sup>1</sup>We run the seismic modeling on a GPU from the Santos Dumont cluster at the National Scientific Computing Laboratory at Petropolis/RJ. Details concerning the numerical implementation will be better explored in the next chapters.

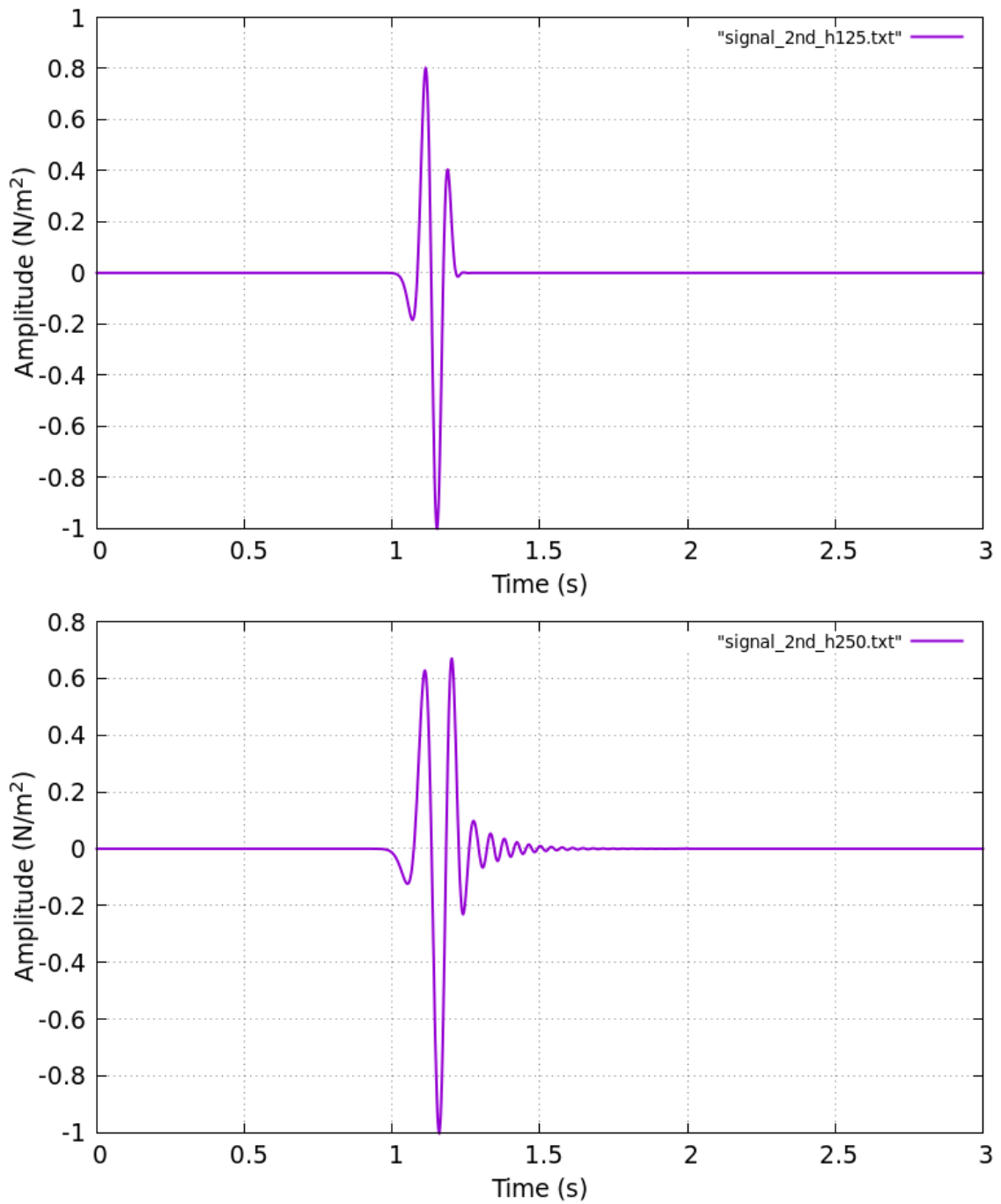


Figure 3.2: Seismic signal for the discrete wave equation that is 2nd-order accurate in time and space. The upper image considers a grid with  $801 \times 801 \times 361$  points, where the grid space is 12.5 meters, and the lower image considers a grid with  $401 \times 401 \times 181$  points, where the grid space is 25.0 meters.

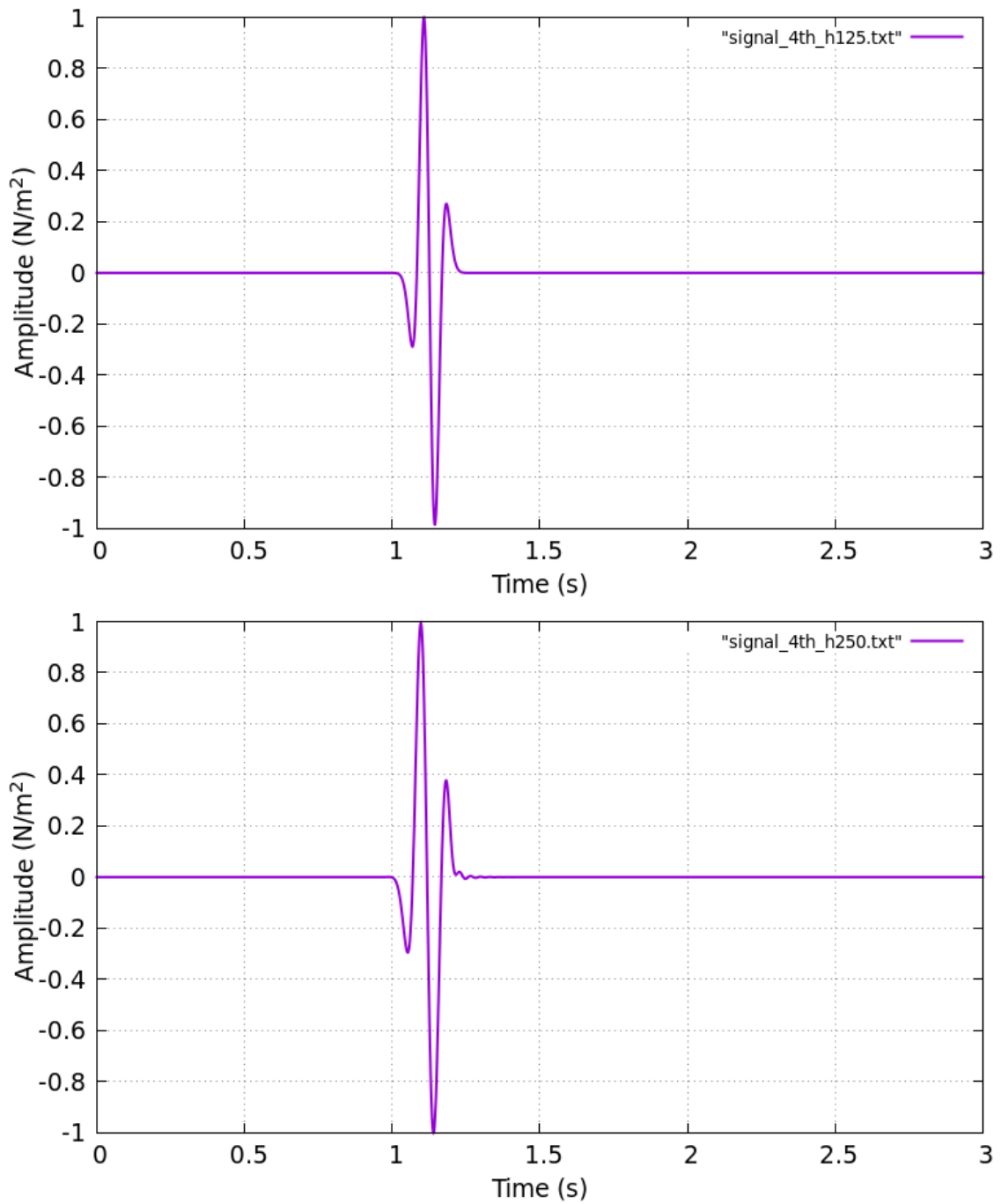


Figure 3.3: Seismic signal for the discrete wave equation that is 2nd-order accurate in time and 4th-order accurate in space. The upper image considers a grid with  $801 \times 801 \times 361$  points, where the grid space is 12.5 meters, and the lower image considers a grid with  $401 \times 401 \times 181$  points, where the grid space is 25.0 meters.

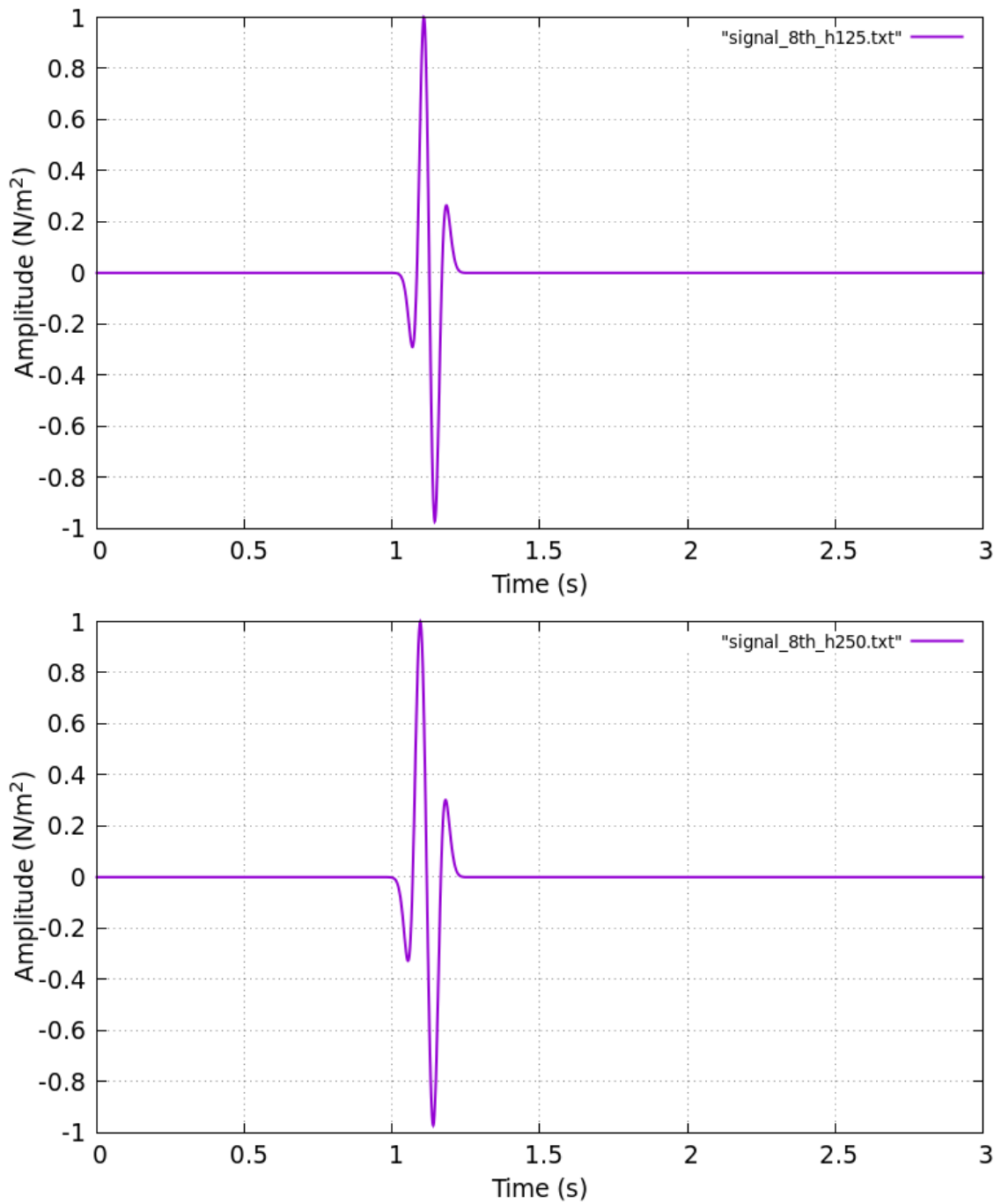


Figure 3.4: Seismic signal for the discrete wave equation that is 2nd-order accurate in time and 8th-order accurate in space. The upper image considers a grid with  $801 \times 801 \times 361$  points, where the grid space is 12.5 meters, and the lower image considers a grid with  $401 \times 401 \times 181$  points, where the grid space is 25.0 meters.

## 3.2 Data compression for wavefield storage

Algorithm 1 detailed in section 3.1 for the wavefield propagation is the kernel of the RTM technique. Algorithm 2 extends the seismic modeling to the RTM implementation. RTM needs as inputs a velocity field, a seismic source, and a set of seismograms,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$  that contains information about the medium reflectivity. The computation of the imaging condition uses, among others, the source, and backward-propagated wavefield (receiver wavefield) solutions to build the migrated seismic section that stacks the partial results over time  $\left(\mathbf{I}_{\sum n_\tau}^k\right)$ , and over the number of seismograms  $\left(\mathbf{I}_{\sum shot\_id}^k\right)$ . The sum in line 19 approximates the time integrals of the seismic imaging equation 2.8. It is possible to derive an equivalent expression for the imaging condition 2.7. To see that, let's consider, for simplicity, the homogeneous acoustic wave equation for a constant slowness  $\sigma$ :

$$\nabla^2 p(\mathbf{r}, t) - \sigma^2 \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} = 0. \quad (3.14)$$

The discrete representations for the Laplacian operator and second-order time derivative following the finite difference equations 3.1, 3.2, 3.3, and 3.4 yield  $\mathbf{L}$ , a symmetric and positive definite matrix, and  $\frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \Big|_{n_t}$ . Thus, the discrete homogeneous acoustic wave equation is:

$$\frac{1}{\sigma^2} \mathbf{L} \mathbf{p} = \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \Big|_{n_t}. \quad (3.15)$$

where  $\mathbf{p}$  is the discrete vector corresponding to  $p(\mathbf{r}, t)$ , and the right-hand side the discrete 2nd-order time derivative.

Substituting equation 3.15 in the generalized imaging condition 2.7 and discretizing  $\bar{p}(\mathbf{r}, t)$  to  $\bar{\mathbf{p}}$  yield the discrete imaging condition  $\mathbf{I}_S^k$  for the  $k$ -th component:

$$\begin{aligned} \mathbf{I}_S^k &= \frac{2}{\sigma} \frac{\sum_{n_\tau=0}^{N_t-1} \bar{\mathbf{p}}_{n_\tau}^k \cdot \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \Big|_{n_t}^k \Delta t}{\sum_{n_t=0}^{N_t-1} \left( \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \Big|_{n_t}^k \cdot \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} \Big|_{n_t}^k \right) \Delta t}, \\ &= \frac{2}{\sigma} \frac{\sum_{n_\tau=0}^{N_t-1} \bar{\mathbf{p}}_{n_\tau}^k \cdot \frac{1}{\sigma^2} [\mathbf{L} \mathbf{p}_{n_t}]^k}{\sum_{n_t=0}^{N_t-1} \left( \frac{1}{\sigma^2} \right)^2 [\mathbf{L} \mathbf{p}_{n_t}]^k \cdot [\mathbf{L} \mathbf{p}_{n_t}]^k}, \\ &= 2\sigma \frac{\sum_{n_\tau=0}^{N_t-1} \bar{\mathbf{p}}_{n_\tau}^k \cdot [\mathbf{L} \mathbf{p}_{n_t}]^k}{\sum_{n_t=0}^{N_t-1} [\mathbf{L} \mathbf{p}_{n_t}]^k \cdot [\mathbf{L} \mathbf{p}_{n_t}]^k}, \end{aligned} \quad (3.16)$$

where  $\cdot$  and  $\dot{}$  represent the component-wise multiplication and division operations for the  $k$ -th component of a vector, and  $n_\tau = N_t - n_t$  represents the discrete reversal time.

Discretizing the imaging condition from equation 2.8 yields:

$$\mathbf{I}_C^k = \frac{\sum_{n_t=0}^{N_t-1} \bar{\mathbf{p}}_{n_t}^k \cdot \mathbf{p}_{n_t}^k}{\sum_{n_t=0}^{N_t-1} \mathbf{p}_{n_t}^k \cdot \mathbf{p}_{n_t}^k}, \quad (3.17)$$

The discrete imaging condition from equation 3.17 is similar to the one in equation 3.16 if  $\mathbf{L} = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix. In this case, the amplitudes of the images generated by equations 3.17 and 3.16 will differ by the scaling factor  $2\sigma$ . However, in the general case both images will differ only in the magnitude of their amplitudes. Therefore, we choose to implement the imaging condition 3.17. To reach that, the source wavefield is computed by solving the wave equation with the independent term being the seismic source and storing it in disk for further access (step 10 in red). On the other hand, the recorded seismograms induce the computation of the receiver wavefield. At the end of Algorithm 2 (line 21), we obtain the discrete seismic image  $I_C \in \mathbb{R}^{N_x \times N_y \times N_z}$ , where the amplitude variations represent physical properties changes.

The implementation presented in Algorithm 2 is one of the simplest ways to build the convolutional imaging condition, but it suffers from a high demand for persistent storage. Storing and accessing the source wavefield into and from the hard disk is challenging for 3D scenarios. For instance, the disk requirements to store the forward-wavefield propagation for the 3D case is  $4 \times N_{shots} \times N_t \times N$  Bytes, where the value 4 stands for a single precision representation of a floating point number. Considering a hypothetical scenario of the wavefield propagation in a grid of  $200 \times 200 \times 200$  grid points spaced of 25.0 meters during 6.0 seconds step-wised of 0.5 milliseconds leads to a disk storage demand of  $\approx 178.81$  GB per shot. Executing the same example in parallel considering 20 shots elevates the disk requirements to 3.5 TB. The hypothetical scenario is far away from real cases like the Iracema and Tupi fields from Santos Basin, where the seismic volumes have dimensions of  $[30.0 \times 30.0 \times 10.0]$  km and  $[42.0 \times 45.0 \times 10.0]$  km, respectively [2, 110]. In these cases, the number of grid points can reach  $1201 \times 1201 \times 401$  and  $1681 \times 1801 \times 401$  for the Iracema and Tupi fields considering a grid space of 25.0 meters. Assuming an acquisition time of 6.0 seconds step-wised of 0.5 milliseconds the disk requirements to store the wavefield are 2.53 TB and 5.43 TB per shot, respectively.

Data compression presents itself as an efficient strategy to reduce persistent storage in seismic migration algorithms like RTM [9, 17, 65, 66, 76, 112, 127]. Compressing the data consists of converting an input data stream into another one that has a smaller size, where the stream can be a file on disk or a buffer on memory. The conversion process reduces or removes redundancies from the original data by exploiting a smaller representation of the data [65, 75, 112]. There are two classes of methods for data compression, which are the lossy and lossless algorithms. If the numerical values are only approximately represented after they have been decompressed, the compression is called lossy.

---

**Algorithm 2** Reverse Time Migration

---

**Require:**  $\mathbf{v}$ ,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , and  $\mathbf{f}$ 

```
1: function RTM( vector  $\mathbf{v}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
2:   read  $\mathbf{v}$ , and  $\mathbf{f}$ 
3:   initialize image condition  $\mathbf{I}_{\Sigma shot\_id} = 0$ 
4:   for  $shot\_id = 1$  to  $N_{shots}$  do
5:     initialize  $n_t = 0$ 
6:     apply initial conditions for  $i_t = 0$ 
7:     for  $i_t = 1$  to  $N_t$  do
8:        $n_t = i_t * \Delta t$ 
9:       solve equation (2.2) ▷ source wavefield
10:      store  $\mathbf{p}_{n_t}$  for all  $n_t$ 
11:    end for
12:    initialize  $n_\tau = 0$ , and  $\mathbf{I}_{\Sigma \tau} = 0$ 
13:    apply initial conditions for  $i_\tau = 0$ 
14:    read  $\mathbf{s}_{shot\_id}$ 
15:    for  $i_\tau = 1$  to  $N_t$  do
16:       $n_\tau = (N_t - i_\tau) * \Delta \tau$  ▷ reverse time
17:      read  $\mathbf{p}_{n_\tau}$ 
18:      solve equation (2.5) ▷ receiver wavefield
19:      calculate  $\mathbf{I}_{\Sigma n_\tau}^k = \mathbf{I}_{\Sigma n_\tau}^k + (\mathbf{p}_{n_\tau}^k \cdot \bar{\mathbf{p}}_{n_\tau}^k) / (\mathbf{p}_{n_\tau}^k \cdot \mathbf{p}_{n_\tau}^k)$  ▷ imaging condition
20:    end for
21:    stack  $\mathbf{I}_{\Sigma shot\_id}^k = \mathbf{I}_{\Sigma shot\_id}^k + \mathbf{I}_{\Sigma n_\tau}^k$  ▷ stacking
22:  end for
23:   $\mathbf{I}_C \leftarrow \mathbf{I}_{\Sigma shot\_id}^k$ 
24:  store  $\mathbf{I}_C$ 
25: end function
```

---

Otherwise, if the values are represented exactly as the original data, it is called lossless compression [75, 76].

There are several lossy and lossless compressors such as Gzip [37], DEFLATE algorithm [45], ZLIB [49], SZ library [38], MGARD library [5], TuckerMPI [116] among others. We have chosen the ZFP compressor<sup>2</sup>, which is an open-source C/C++ library for compressing integer and floating-point data of multidimensional numerical arrays. The ZFP compressor [77] operates on  $d$ -dimensional arrays ( $d \in [1, 4]$ ) by partitioning them into blocks of  $4^d$  values. The blocks are independently compressed/decompressed from each other. Because of that, ZFP is recognized as one of the most effective high-speed lossy floating-point compressors. To achieve high compression rates, ZFP uses lossy but optionally error-bounded compression controlled by four different parameters known as expert, fixed-rate, fixed-precision, and fixed-accuracy modes. On the other hand, although bit-for-bit lossless compression of floating-point data is not always possible, ZFP also offers an accurate near-lossless compression. For that, we can use the reversible mode [40, 75].

---

<sup>2</sup><https://computing.llnl.gov/projects/zfp>

We employ the ZFP fixed-accuracy mode to compress the 4-D numerical array that stores the source wavefield  $\mathbf{p}_{n_t}$  (line 10 of Algorithm 2). According to ZFP documentation<sup>3</sup>, the fixed-accuracy mode guarantees that each reconstruct value falls within a user-specified absolute error tolerance, which usually results in the smallest RMS error and therefore highest peak signal to noise ratio for the same rate. The ZFP library can be linked to Algorithm 2 as a static/dynamic link for C/C++ language implementation, for instance. Thus, we modify the RTM algorithm aiming to compress the source wavefield. Besides, instead of storing the source wavefield for every time step ( $\Delta t$ ) based on the FDM, we took advantage of the Nyquist theory as explored by SUN and FU [112] to store the wavefield at the Nyquist time step to reduce the amount of information. The Nyquist time step  $\Delta t_{nyq}$  is defined as,

$$\Delta t_{nyq} = \frac{1}{2(f_{max} - f_{min})}, \quad (3.18)$$

where  $f_{max}$  and  $f_{min}$  are the highest and lowest frequency of the seismic source.

Algorithm 3 details the RTM algorithm, which takes into account the compression of the source wavefield re-sampled by the Nyquist time step. Most implementations follow the same strategy of Algorithm 2, but we modify lines 10 and 16 to consider the source wavefield compression. Thus, lines 10 and 11 in Algorithm 3 show the compression of the multidimensional array  $\mathbf{p}$  for the Nyquist time step  $\Delta t_{nyq}$ , and its storage on disk. Lines 18 and 19 describe the reading of  $\mathbf{p}$  also for  $\Delta t_{nyq}$  and its decompression. As mentioned earlier, we use the ZFP library for compression/decompression and the Nyquist theory to calculate the value of  $\Delta t_{nyq}$ .

### 3.3 Wavefield reconstruction

The RTM implementation as presented in Algorithm 2, as well as in Algorithm 3, suffers from persistent I/O due to the need of storing the source wavefield in disk for further access to calculate the imaging condition. However, we can use the Nyquist theory to guide the frequency of the wavefield storage based on the physical properties of the medium. This approach personalizes the decimation process to a storage frequency that can be higher or lower than the default decimation factor ( $s = 10$  as we discuss in Appendix B). Besides, we can apply the compression to the source wavefield before its storage to reduce even more the amount of information on the disk. Although the compression linked to decimation based on the Nyquist theory reduces persistent storage, the application for large scales of frequencies up to 20.0 Hz is still challenging [103].

Another way to overcome this issue, explored in this work, is to reconstruct the source wavefield from information generated during the first part of the RTM, that is, forward

---

<sup>3</sup><https://zfp.readthedocs.io/en/release0.5.5/modes.html#fixed-accuracy-mode>

---

**Algorithm 3** Reverse Time Migration with Wavefield Compression
 

---

**Require:**  $\mathbf{v}$ ,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , and  $\mathbf{f}$ 

```

1: function RTM( vector  $\mathbf{v}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
2:   read  $\mathbf{v}$ ,  $\mathbf{f}$ , and  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ 
3:   initialize image condition  $\mathbf{I}_{\Sigma shot\_id} = 0$ 
4:   for  $shot\_id = 1$  to  $N_{shots}$  do
5:     initialize  $n_t = 0$ 
6:     apply initial conditions for  $i_t = 0$ 
7:     for  $i_t = 1$  to  $N_t$  do
8:        $n_t = i_t * \Delta t$ 
9:       solve equation (2.2) ▷ source wavefield
10:      compress  $\mathbf{p}_{n_t}$  for  $n_t \mid \text{mod}(n_t, \Delta t_{nyq}) = 0$ 
11:      store the compressed  $\mathbf{p}_{n_t}$ 
12:    end for
13:    initialize  $n_\tau = 0$ , and  $\mathbf{I}_{\Sigma \tau} = 0$ 
14:    apply initial conditions for  $i_\tau = 0$ 
15:    for  $i_\tau = 1$  to  $N_t$  do
16:       $n_\tau = (N_t - i_\tau) * \Delta \tau$  ▷ reverse time
17:      read  $\mathbf{s}_{shot\_id}$ 
18:      read compressed  $\mathbf{p}_{n_\tau}$  for  $n_\tau \mid \text{mod}(n_\tau, \Delta t_{nyq}) = 0$ 
19:      decompress  $\mathbf{p}_{n_\tau}$ 
20:      solve equation (2.5) ▷ receiver wavefield
21:      calculate  $\mathbf{I}_{\Sigma n_\tau}^k = \mathbf{I}_{\Sigma n_\tau}^k + (\mathbf{p}_{n_\tau}^k \cdot \bar{\mathbf{p}}_{n_\tau}^k) / (\mathbf{p}_{n_\tau}^k \cdot \mathbf{p}_{n_\tau}^k)$  ▷ imaging condition
22:    end for
23:    stack  $\mathbf{I}_{\Sigma shot\_id}^k = \mathbf{I}_{\Sigma shot\_id}^k + \mathbf{I}_{\Sigma n_\tau}^k$  ▷ stacking
24:  end for
25:   $\mathbf{I}_C \leftarrow \mathbf{I}_{\Sigma shot\_id}^k$ 
26:  store  $\mathbf{I}_C$ 
27: end function

```

---

wave propagation [93]. To reconstruct the source wavefield, we implement the Initial Value Reconstruction (IVR) methodology first explored by FARIA [44] and SYMES [113]. The IVR proposed by SYMES [113] stores temporary states of the wavefield known as checkpoints. Such states are used later for recursive recomputations of the source wavefield. On the other hand, FARIA [44] uses a single checkpoint to initiate the backpropagation of the wavefield. However, using this concept with non-reflective boundary conditions can result in an inefficient reconstruction of the source wavefield due to signal attenuation in the boundary [109]. But, the complete reconstruction of the wavefield can be achieved by keeping all energy in the system and solving the following reconstructed wave equation:

$$\nabla^2 p^R(\mathbf{r}, \tau) - \sigma(\mathbf{r})^2 \frac{\partial^2 p^R(\mathbf{r}, \tau)}{\partial \tau^2} = f(\tau) \delta(\mathbf{r}_s - \mathbf{r}), \quad (3.19)$$

where  $p^R$  is the reconstructed source wavefield defined in  $\Omega \subset \mathbb{R}^{nd}$ . Boundary conditions

can be set as equations 2.2 and 2.5, that is  $p^R(\mathbf{r}, \tau) = 0$  on  $\partial\Omega$ . Lastly, the initial conditions are set as  $p^R(\mathbf{r}, 0) = p(\mathbf{r}, T)$ , and  $\partial^2 p^R(\mathbf{r}, 0) / \partial \tau^2 = \partial^2 p(\mathbf{r}, T) / \partial t^2$  after solving equation 2.2,  $f(0)\delta(\mathbf{r}_s - \mathbf{r}) = f(T)\delta(\mathbf{r}_s - \mathbf{r})$ , and  $\tau = T - t$  is the reversal time. Appendix C shows the equivalence between the source and reconstructed wavefield solutions.

Unwanted signals come from the boundary due to the absence of attenuated layers on the boundaries used to simulate truncated domains. One way to overcome this issue is generating random signals coming from the boundaries induced by randomized velocities on the boundaries as explored in CLAPP [32]. This approach known as Random Boundary Condition (RBC) is based on the idea that what matters for the calculation of the RTM imaging condition is the coherent reflections coming from the boundaries [32]. Thus, CLAPP [32] proposed to introduce a random component to the velocity field at the boundaries. Notice that the random velocity field has to respect the numerical stability constraint of the FDM. It is expected that the random source wavefield coming from the boundaries does not coherently correlate with the receiver wavefield.

Furthermore, a smoother transition from the inner domain to the boundaries is ideal. The smooth transition will avoid unwanted immediate reflections of the randomized area. One way to build a smooth transition area is by multiplying coefficients  $c_i$  to the random vector velocity  $\mathbf{v}$  in the normal direction to the boundaries, where the index  $i \in [1, \dots, N_a]$  where  $N_a$  is positive real number which represents the size of the additional region related to the boundaries (size thickness). The coefficients are responsible for slowing down the velocities values, and SILVA [109] showed that values between the linear and Gaussian functions, represented by equations equation (3.20) and equation (3.21), build the best coefficients, that is,

$$g(x) = (N_a - x) \left( \frac{1}{N_a - 1} \right), \quad (3.20)$$

$$h(x) = \exp \left( -120(x - 1)^2 \left( \frac{1}{N_a - 1} \right)^2 \right), \quad (3.21)$$

where,  $x \in [1, N_a]$ . Let  $\mathbf{g}$ , and  $\mathbf{h}$  be the discrete version of the functions  $g(x)$ , and  $h(x)$  after numerical discretization, thus the damping coefficients assume values in  $\mathbf{h} \leq c_i \leq \mathbf{g}$  for  $i \in [1, \dots, N_a]$ .

Algorithm 4 highlights in blue the main modifications in the basic RTM algorithm. We use vector  $\mathbf{p}^R$  to represent the finite difference discretization of equation 3.19. The first part of the RTM with wavefield reconstruction calculates the source wavefield, and the last two moments of the wavefield are stored (line 11). After reading the stored wavefield moments, the second part of the algorithm that calculates the receiver wavefield also calculates the reconstruction of the source wavefield  $\mathbf{p}^R$  by solving equation 3.19. Thus, the modified algorithm stores only two panels of source wavefield instead of all panels for each  $n_t$ . This strategy comes with the additional cost of solving one extra wave equation.

---

**Algorithm 4** Reverse Time Migration with Wavefield Reconstruction
 

---

**Require:**  $\mathbf{v}$ ,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , and  $\mathbf{f}$

```

1: function RTM( vector  $\mathbf{v}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
2:   read  $\mathbf{v}$ ,  $\mathbf{f}$ , and  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ 
3:   create a RBC as Algorithm 2 from CLAPP [32]
4:   initialize image condition  $\mathbf{I}_{\Sigma shot\_id} = 0$ 
5:   for  $shot\_id = 1$  to  $N_{shots}$  do
6:     initialize  $n_t = 0$ 
7:     apply initial conditions for  $i_t = 0$ 
8:     for  $i_t = 1$  to  $N_t$  do
9:        $n_t = n_t + i_t * \Delta t$ 
10:      solve equation (2.2) ▷ source wavefield
11:      store  $\mathbf{p}_{n_t}$  for  $N_{t-1}$ , and  $N_t$ 
12:    end for
13:    initialize  $n_\tau = 0$ , and  $\mathbf{I}_{\Sigma \tau} = 0$ 
14:    read  $\mathbf{p}_{N_t}$ , and  $\mathbf{p}_{N_{t-1}}$ 
15:    apply initial conditions for  $i_\tau = 0$ 
16:    for  $i_\tau = 1$  to  $N_t$  do
17:       $n_\tau = N_t - (n_\tau + i_\tau * \Delta \tau)$  ▷ reverse time
18:      read  $\mathbf{s}_{shot\_id}$ 
19:      solve equation (2.5) ▷ receiver wavefield
20:      solve equation (3.19) ▷ wavefield reconstruction
21:      calculate  $\mathbf{I}_{\Sigma n_\tau}^k = \mathbf{I}_{\Sigma n_\tau}^k + ([\mathbf{p}_{n_\tau}^R]^k \cdot [\bar{\mathbf{p}}_{n_\tau}^R]^k) / ([\mathbf{p}_{n_\tau}^R]^k \cdot [\mathbf{p}_{n_\tau}^R]^k)$  ▷ imaging
      condition
22:    end for
23:    stack  $\mathbf{I}_{\Sigma shot\_id}^k = \mathbf{I}_{\Sigma shot\_id}^k + \mathbf{I}_{\Sigma n_\tau}^k$  ▷ stacking
24:  end for
25:   $\mathbf{I}_C \leftarrow \mathbf{I}_{\Sigma shot\_id}^k$ 
26:  store  $\mathbf{I}_C$ 
27: end function

```

---

Figure 3.5 shows the representation of the propagation of the forward wavefield (Figures 1(A), 1(B), 1(C), and 1(D)) and its reconstruction (Figures 1(E), 1(F), 1(G), and 1(H)) in a constant velocity field of 2000 m/s based on Algorithm 4. The experiment from Figure 3.5 uses the RBC, and thus, it is possible to see the incoherent signals coming from the boundaries. Due to its nature, the random wavefield tends to disappear over the stacking process of the RTM algorithm, and such reflections coming from the boundaries do not hamper the final seismic image [9, 31, 72]. Besides, the computational implementation for the Algorithm 4 maintains all energy inside the domain, allowing the complete reconstruction of the source wavefield.

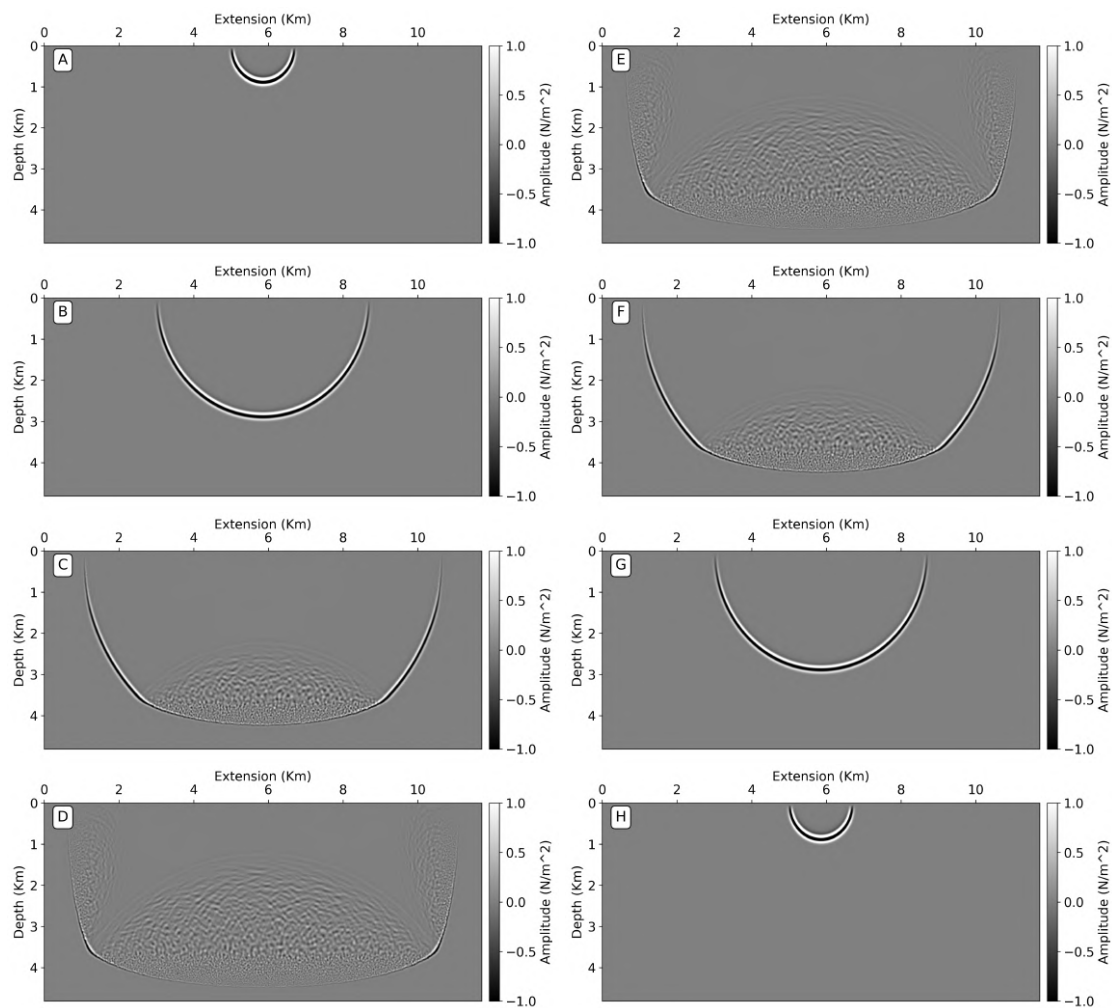


Figure 3.5: The left column shows the representation of the source wavefield for the instants 0.25s (A), 1.5s (B), 2.5s (C), and 3.0s (D). The right column shows the representation of the wavefield reconstruction of the source wavefield for instants 0.25s (E), 1.5s (F), 2.5s (G), and 3.0s (H). The results were provided by Algorithm 4.

# Chapter 4

## Computational Implementation

We present in this chapter the computational implementations for different platforms, such as multi-CPU, vector processors, and multi-GPU machines. Different from Chapter 3, where we present the RTM algorithms with wavefield storage and reconstruction, we present in this chapter the C language implementations for the key parts of the RTM algorithms on three different computational environments. Thus, we use the OpenMP library for multi-CPU machines, automatic parallelization based on compilation flags for vector processor platforms, and the OpenACC library for multi-GPU machines. The description of each one is discussed in sections 4.1, 4.2, and 4.3, respectively. Lastly, section 4.4 details the MPI implementation to handle multiple shots on multiple nodes.

### 4.1 Optimized multi-core baseline implementation

We develop a version for RTM on a traditional scalar multi-CPU machine to obtain a parallel optimized baseline implementation. As mentioned in Chapter 3, the RTM is composed of two parts. The first one calculates the source wavefield and the second part calculates the receiver wavefield. Our baseline implementation is written in C programming language, and it takes advantage of OpenMP directives to explore multiple cores/threads parallelism. These are two of the different level parallelisms implemented in our applications. Another one takes advantage of the single-instruction-multiple-data (SIMD) model. Thus, depending on where we compile the applications, the SSE or AVX instructions (for Intel compilers) activate vectorization. This is known as data-level parallelism (see details in Appendix D). The multi-core/thread parallelism and vectorization can be applied to the second-order wave equation that is discretized with a second-order finite difference scheme in time and an eighth-order scheme in space (equation 3.9). The stencil presented in equation 3.9 is the basis for the source and receiver wavefield calculations.

Listing 4.1 details the nested loops (lines 13, 14, and 16) to calculate the source wavefield over the temporal loop represented in line 6 based on the wave equation stencil.

For that, allocations for the source wavefield, seismic wavelet, and auxiliary variables are needed. Besides, the computational implementation indicates that the source wavefield is obtained for a set of shots (line 3). Line 12 shows the OpenMP directives used to parallelize the nested loops of the wave equation stencil. The instruction `omp parallel for` creates a parallel region that is executed by a team of OpenMP threads following the loop construct `for`. Line 15 shows the `omp simd` instruction that activates vectorization and allows operating on multiple data with a single instruction. Concerning the source wavefield, its storage depends if Algorithm 2, 3 or 4 is implemented.

```

1 // Declarations and allocations
2   ...
3 // Starts rtm loop per shot
4 // ...
5
6 for (time = 0; time < numberOfTimeStep; time++) {
7
8     // Insert seismic source
9     // ...
10
11    // Nested loops of the forward wave propagation
12    #pragma omp parallel for private(i,j,k)
13    for (j = HALFORDERDF; j < dimY-HALFORDERDF; j++) {
14        for (i = HALFORDERDF; i < dimX-HALFORDERDF; i++) {
15            #pragma omp simd
16            for (k = HALFORDERDF; k < dimZ-HALFORDERDF; k++) {
17
18                }
19            }
20        }
21
22    // Record and update source wavefield
23    // ...
24    }
25
26 // End rtm loop per shot

```

Listing 4.1: Optimized baseline implementation applied to the wave equation stencil.

Reversal time implementation of the RTM explores the technique proposed by GIVOLI [52]. This technique guarantees the coercivity of the adjoint problem. Listing 4.2 details the backward wavefield propagation to calculate the receiver wavefield. The reversal time wavefield propagation is calculated after the forward wavefield propagation by inserting the seismic signals of the seismogram. The insertion of the seismogram uses the variable changing  $\tau = T - t$  (lines 10 and 19). Similar to Listing 4.1, we use the OpenMP directives `pragma omp parallel for` and `pragma omp simd` to parallelize the nested loops of the seismogram insertion (lines 14 and 16) and backward wavefield

propagation (lines 25, 26, and 28).

```
1 // Declarations and allocations
2 ...
3 // Starts rtm loop per shot
4 // ...
5 // Forward-wavefield propagation
6 // ...
7
8 for (tau = 0; tau < numberOfTimeStep; tau++) {
9
10     time = numberOfTimeStep - tau - 1;
11
12     // Insert seismogram
13     #pragma omp for private(irec,j,rx,ry)
14     for (j = 0; j < numberOfStreamers; j++) {
15         #pragma omp simd
16         for (irec = 0; irec < numberOfReceivers; irec++) {
17             // receiver coordinates
18             ...
19             receiver_wavefield01 [...] = - seismogram[time + ...];
20         }
21     }
22
23     // Nested loops of the backward wavefield propagation
24     #pragma omp parallel for private(i,j,k)
25     for (j = HALFORDERDF; j < dimY-HALFORDERDF; j++) {
26         for (i = HALFORDERDF; i < dimX-HALFORDERDF; i++) {
27             #pragma omp simd
28             for (k = HALFORDERDF; k < dimZ-HALFORDERDF; k++) {
29
30                 }
31         }
32     }
33
34     // Read source wavefield or source wavefield reconstruction
35     // ...
36     // Calculate imaging condition and update receiver wavefield
37     // ...
38     }
39
40 // End rtm loop per shot
```

Listing 4.2: Optimized baseline implementation applied to the reversal time wave equation stencil.

The imaging condition can be built during the receiver wavefield calculation and the reading of the source wavefield from disk, as detailed in Algorithm 2. Thus, the process

avoids explicit storage of the receiver wavefield in the disk as performed for the source wavefield. On the other hand, access to the source wavefield can also be achieved through wavefield reconstruction as detailed in Algorithm 4. Notice that every nested loop of the RTM algorithm can be parallelized using the OpenMP directives presented in Listing 4.1 and 4.2.

## 4.2 Vector processor parallelism implementation

The computational vector processor implementation is directed toward the NEC SX-Aurora TSUBASA type 10B vector processor. As better detailed in Appendix D.1, the SX-Aurora TSUBASA architecture consists of a vector engine (VE) equipped with a vector processor and a vector host (VH). In this architecture, the VE runs the entire application, while the VH is responsible for processing system calls invoked by the application [64]. Besides, the architecture avoids frequent data transfers between the VE and its VH. Developing scientific applications for the SX-Aurora TSUBASA is straightforward because no special coding is required, and the developers do not need to take care of system calls.

The NEC SX-Aurora TSUBASA provides support for OpenMP implementations as well as automatic parallelization and vectorization. Both can be activated by NEC flag compilations. Instead of using explicit parallelization for the NEC vector processor, we have been using the automatic parallelization and vectorization activated by the NEC compilation flags `-O4`, `mparallel`, `-fivdep`, and `-mparallel-innerloop`. The `-O4` flag defines the level of optimization 4, the `mparallel` allows automatic parallelization, `-fivdep` inserts `ivdep` directive before all loops for vectorization, and `-mparallel-innerloop` allows to parallelize the inner-loop. Thus, we omit the parallel OpenMP solutions presented in Listing 4.1 and 4.2 for the nested loops using instead the NEC compilation flags. See the online guide<sup>1</sup> for more details of the NEC compilation flags. Last but not least, we use the `proginf` and `ftrace` tools to analyze the computational performance of our automatic optimization choices (see Appendix D.1). According to the profile files and better detailed in the numerical experiments (Chapter 5), the computational optimizations produced high vector operation ratios and extremely low cache missing for both seismic modeling and RTM applications.

## 4.3 GPU parallelism implementation

The GPU programming model, based on OpenACC directives<sup>2</sup>, aims to provide an easier way for scientific applications coding [67, 100]. Besides, compared to CUDA,

---

<sup>1</sup><https://sxaoratsubasa.sakura.ne.jp/documents/sdk/pdfs/g2af01e-C++UsersGuide-027.pdf>

<sup>2</sup>See Appendix E for a quick start on OpenACC.

OpenACC programming demands less coding efforts in heterogeneous environments with CPU+GPU [100, 105]. The OpenACC implementation needs to deal with three main issues: CPU (host) calculations, GPU calculations, and communications to and from the GPU. Thus, any computational implementation must maximize the GPU computations and prevent communications between the host and GPU.

Algorithm 5 details the host and GPU calculations and the communication between them for the seismic modeling, which simulates the wavefield propagation. It is worth mentioning that seismic modeling is the RTM kernel, and we will present it first. In seismic modeling, the first operations made by the host are data allocation followed by disk reading and storage of the velocity field and seismic source information in the vectors  $\mathbf{v}$ , and  $\mathbf{f}$ . These steps are shown in lines 2 and 3 in Algorithm 5. Following, the main data, such as the vectors  $\mathbf{v}$  and  $\mathbf{f}$ , are moved to the GPU (line 4). Lines 6 and 7 show the GPU operations for the wave equation calculation once the necessary information is transferred and allocated. The seismogram is the outcome of the wave equation simulation, and it is transferred from GPU to the host in line 9. The final operations of the host are seismogram storage and data deallocation in lines 10 and 11. Notice that for seismic modeling, only three communications are necessary. Because the velocity field and seismic source are information provided for the seismic modeling, two transfers are made from the host to GPU. In the end, the host stores the seismogram after its transfer from GPU to the host.

---

**Algorithm 5** Seismic Modeling GPU Implementation

---

**Require:**  $\mathbf{v}$ , and  $\mathbf{f}$

```

1: function SEISMIC_MODELING_GPU( vector  $\mathbf{v}$ , vector  $\mathbf{f}$  )
2:   allocate data variables                                     ▷ Host computations
3:   read  $\mathbf{v}$ , and  $\mathbf{f}$ 
4:   Move data to GPU                                         ▷ Data transfer and allocation
5:   for time = first to last do
6:     solve wave equation (2.2)                               ▷ GPU computations
7:     record seismogram
8:   end for
9:   update host with seismogram                               ▷ Data transfer and deallocation
10:  store seismogram                                         ▷ Host computations
11:  deallocate data variables
12: end function

```

---

We use the ACC DATA COPYIN directive for transferring the data from the host to GPU. ACC DATA COPYOUT directive transfers the data from GPU to host. ACC DATA CREATE allocates necessary vectors in the GPU. For parallelization, we use the ACC LOOP directive. Listing 4.3 shows the OpenACC implementation for wavefield propagation. Lines 6 to 13 show the copyin, copyout, and create pragmas for data transfer and allocation on the GPU. The kernels directive generates by compiler analysis parallel regions (or kernels). In our case, the parallel region is represented by the loop

pragma in line 23. The independent instruction tells the compiler that iterations are independent. Lastly, the collapse instruction indicates that the nested loops are treated as a single one.

```

1 // Declarations and allocations
2   ...
3 // Starts wavefield propagation
4 // ...
5
6 #pragma acc data copyin(coef[0:dimX*dimY*dimZ])
7 #pragma acc data copyin(wavelet[0:sampligWavelet])
8
9 #pragma acc data create(point_aux[0:dimX*dimY*dimZ])
10 #pragma acc data create(wavefield01[0:dimX*dimY*dimZ])
11 #pragma acc data create(wavefield02[0:dimX*dimY*dimZ])
12
13 #pragma acc data copyout(seismogram[0:numberOfReceivers*
    numberOfCables*numberOfSamples])
14
15 for (time = 0; time < numberOfTimeStep; time++) {
16     #pragma acc kernels
17     {
18
19         // Insert seismic source
20         // ...
21
22         // Nested loops of the forward wave propagation
23         #pragma acc loop independent collapse(3)
24         for (j = HALFORDERDF; j < dimY-HALFORDERDF; j++) {
25             for (i = HALFORDERDF; i < dimX-HALFORDERDF; i++) {
26                 for (k = HALFORDERDF; k < dimZ-HALFORDERDF; k++) {
27
28                     }
29                 }
30             }
31
32             // Record seismogram and update source wavefield
33             // ...
34         }
35     }
36
37 // End wavefield propagation

```

Listing 4.3: OpenACC implementation for the wave equation stencil.

Most of the OpenACC implementations presented in Algorithm 5 and Listing 4.3 for the seismic modeling are the same for the RTM algorithm. The differences between seismic modeling and RTM algorithms are mainly related to data transfer. For instance,

Algorithm 6 details the OpenACC implementation for the RTM which implements the wavefield storage (Algorithm 2). Again, we use three different colors to represent the host computations (green), data transfer (blue), and GPU calculations (red). The first part of Algorithm 6 moves the source wavefield during its calculation from the GPU to the host and stores it in disk (lines 6 to 9). In general, storing the source wavefield in a disk is needed because the GPU memory or RAM is insufficient to store it. The second part of the RTM algorithm (lines 13 to 19) moves back the source wavefield from the host to GPU, calculates the receiver wavefield, and builds the imaging condition (lines 15 to 18). To calculate the receiver wavefield, line 12 moves the seismograms from the disk to the GPU.

Algorithm 6 requires two data transfers for the velocity field and seismic source,  $N_{shots}$  data transfers for the seismograms,  $2 \times N_t$  data transfers for the source wavefield, and  $N_{shots}$  data transfers for the seismic images. As shown in Listing 4.3, we use the `copyin` directive for the velocity field, seismic source, and seismograms, `copyout` directive for the seismic images, `create` directive to allocate the vectors for the wavefield solution. The `kernels loop` directive creates the parallel region and parallelizes the nested loops. Lastly, we use the `update self` directive to move the source wavefield from the GPU and store it in the disk, and `update device` to read the source wavefield from the disk and move it to the GPU. Note that we did not show the C language code for this case because the computational implementation is similar to Listing 4.3.

The OpenACC implementation based on Algorithm 4 is shown in Algorithm 7. Remember that Algorithm 4 implements the wavefield reconstruction, and one extra wave equation is required for that. Because of that, its computational implementation does not fully store the source wavefield, only the last two time-frames. The data transfer based on the OpenACC implementation occurs between the two main stages of the RTM technique and not during the temporal loops as the Algorithm 6. Thus, Algorithm 7 requires only four data transfers between the GPU and host for the source wavefield. Hence, instead of using the `update self` and `update device` directives, we use the `copyin` and `copyout` directives.

---

**Algorithm 6** RTM GPU Implementation based on Algorithm 2

---

**Require:**  $\mathbf{v}$ ,  $\mathbf{f}$ , and  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$

```
1: function RTM_GPU( vector  $\mathbf{v}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
2:   allocate data variables ▷ Host computations
3:   read  $\mathbf{v}$ , and  $\mathbf{f}$ 
4:   Move data to GPU ▷ Data transfer and allocation
5:   for time = first to last do
6:     solve wave equation (2.2) ▷ GPU computations
7:     record source wavefield for every time
8:     move source wavefield to host ▷ Data transfer
9:     store source wavefield for each time ▷ Host computations
10:  end for
11:  read seismograms
12:  Move seismogram to GPU ▷ Data transfer and allocation
13:  for time = first to last do
14:    set reversal time evolution
15:    read source wavefield for each reversal time ▷ Host computations
16:    Move source wavefield to GPU ▷ Data transfer and allocation
17:    solve wave equation (2.5) ▷ GPU computations
18:    calculate imaging condition
19:  end for
20:  update host with seismic image ▷ Data transfer and deallocation
21:  store seismic image ▷ Host computations
22:  deallocate all the data variables
23: end function
```

---

---

**Algorithm 7** RTM GPU Implementation based on Algorithm 4

---

**Require:**  $\mathbf{v}$ ,  $\mathbf{f}$ , and  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$

```
1: function RTM_GPU( vector  $\mathbf{v}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
2:   allocate data variables ▷ Host computations
3:   read  $\mathbf{v}$ , and  $\mathbf{f}$ 
4:   Move data to GPU ▷ Data transfer and allocation
5:   for time = first to last do
6:     solve wave equation (2.2) ▷ GPU computations
7:     record source wavefield for every time
8:   end for
9:   update host with the last two wavefield timeframes ▷ Data transfer and
   deallocation
10:  read seismograms
11:  Move seismogram and wavefield timeframes to GPU ▷ Data transfer and
   allocation
12:  for time = first to last do
13:    set reversal time evolution
14:    solve wave equation (2.5) ▷ GPU computations
15:    solve wave equation (3.19)
16:    calculate imaging condition
17:  end for
18:  update host with seismic image ▷ Data transfer and deallocation
19:  store seismic image ▷ Host computations
20:  deallocate all the data variables
21: end function
```

---

## 4.4 Hybrid parallelism implementation

Sections 4.1, 4.2, and 4.3 show the main steps to manipulate the vector data and to parallelize the nested loop that solves the acoustic wave equations used to calculate the RTM seismic image. Parallelism can be applied in different ways depending on the machine architecture. We use the OpenMP library for CPU machines, automatic parallelization based on compilation directives for vector machines, and the OpenACC library for GPU machines. In this section, we discuss how to use the MPI library to share the workload among CPU nodes, vector processor nodes, and GPU nodes. Our strategy explores the non-dependency among the partial RTM solutions for seismic sources (or shots) at different locations when computational resources for the RTM application can be allocated on the RAM or GPU memory, for instance. If this is not possible, we have to implement a domain decomposition approach to share the computational resources through the computational nodes as described in DE MEDEIROS LARA [36], JORGE [61], LOUBOUTIN et al. [84], QAWASMEH et al. [100], WANG et al. [118] among others.

Listing 4.4 shows the C language RTM implementation to deal with multiple shots on multi-GPU machines. Aiming to simplify the discussion, we will present the MPI implementation only for multi-GPU machines, and later, detail the modifications for CPU and vector processor machines. Lines 5 and 6 are responsible for initializing the MPI execution environment and the runtime for the NVIDIA GPU device (`acc_device_nvidia` argument). Following, the number of MPI processes, their respective rank in the communicator group, and the number of GPU devices are set by the instructions in lines 9, 12, and 15. The instruction `acc_set_device_num` (line 20) assigns one MPI process per GPU after associating the process rank to the number of devices. Thus, it is a general rule to define the number of MPI processes equal to the number of GPUs. Lastly, lines 23, 26, and 27 calculate the workload per node, which in our scenario is the number of shots in the RTM application per node. The variables `first_shot` and `last_shot` are used by the loop in line 29 to control the workload per node. The loop in line 29 iterates under the number of shots and can be executed independently.

```
1 // Declarations and allocations
2   ...
3
4 // Initialize MPI environment...
5 MPI_Init(NULL, NULL);
6 acc_init(acc_device_nvidia);
7
8 // Get the number of processes...
9 MPI_Comm_size(MPI_COMM_WORLD, &world_size);
10
11 // Get the rank of the process...
12 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```

13
14 // get the number of GPUs...
15 num_devices = acc_get_num_devices(acc_device_nvidia);
16
17 device_id = rank % num_devices;
18
19 // assign GPU to one MPI process...
20 acc_set_device_num(device_id, acc_device_nvidia);
21
22 // Number of shot per node...
23 shots_per_node = ( (float)numberOfShots ) / world_size;
24
25 // Shot ranges...
26 first_shot = (int) (ceil((shots_per_node * rank) + 1)) ;
27 last_shot  = (int) (ceil( shots_per_node * (rank + 1)));
28
29 for (shotID = first_shot; shotID <= last_shot; shotID++) {
30
31     // solve forward and backward wavefield problem ...
32 }
33
34 MPI_Finalize();

```

Listing 4.4: MPI implementation for the RTM technique.

The MPI instructions in Listing 4.4 can be modified to run on CPU and vector processor machines. The modifications are the same for both platforms. To reach that, lines 6, 15, 17, and 20, which are OpenACC instructions, should be omitted. The new MPI implementation for CPU and vector processor environments split the workload through the MPI process defined in lines 9 and 12.

# Chapter 5

## Numerical Experiments

We divided the numerical experiments into three parts, where section 5.1 presents the computational performance analysis in terms of execution time, speedup, and storage demand for seismic modeling and RTM on three different computational platforms. Section 5.2 shows the results of the RTM technique, which applies data compression to seismograms and wavefields. We analyze the execution times, overhead, storage demand, and impact of compressing the seismograms and wavefields on the RTM solutions. Lastly, we detail in section 5.3 a wrapped RTM into the MC sampling method that can be used for uncertainty quantification on seismic images. Because RTM is time-consuming and data-intensive, RTM under uncertainty demands even more computational resources. Thus, we compare two implementation strategies on different computational platforms by analyzing their outcomes and total processing time.

### 5.1 Computational performance analysis

In this subsection, we present the performance analysis of the 3D seismic modeling and RTM using three different computational platforms: a CPU cluster, a CPU-GPU cluster, and a vector processor machine. The CPU cluster and CPU-GPU cluster are multicore machines from the Santos Dumont system at the National Scientific Computing Laboratory at Petrópolis/Brazil <sup>1</sup>. The CPU cluster has  $2 \times$  Intel Xeon E5-2695v2 Ivy Bridge processors with 2.4GHZ and 24 cores per node. On the other hand, the CPU-GPU cluster has a  $2 \times$  Intel Xeon Skylake 6252, 2.4GHZ with 48 cores and  $4 \times$  NVIDIA V100 per node. Finally, the vector processor is the NEC SX-Aurora TSUBASA Type 10B with 8 vector cores and VE memory of 48GB <sup>2</sup>. We have chosen the MODEL AF provided by the High Performance Computing for Energy (HPC4E) Seismic Test Suite <sup>3</sup> [35] for the 3D seismic modeling and RTM experiments. The MODEL AF is a model designed as a

---

<sup>1</sup>[https://sdumont.lncc.br/support\\_manual.php?pg=support](https://sdumont.lncc.br/support_manual.php?pg=support)

<sup>2</sup>[https://www.hpc.nec/documents/guide/pdfs/Aurora\\_ISA\\_guide.pdf](https://www.hpc.nec/documents/guide/pdfs/Aurora_ISA_guide.pdf)

<sup>3</sup><https://hpc4e.bsc.es/downloads/hpc-geophysical-simulation-test-suite>

set of 15 layers with constant velocity values and flat topography (Figure 5.1). Besides, the velocity parameter model (velocity field) covers an area of  $10 \times 10 \times 4.5$  km. For all 3D computational performance analysis, we used the Ricker seismic source [117] with 20Hz cutoff frequency, and a total acquisition time of 6.0 seconds.

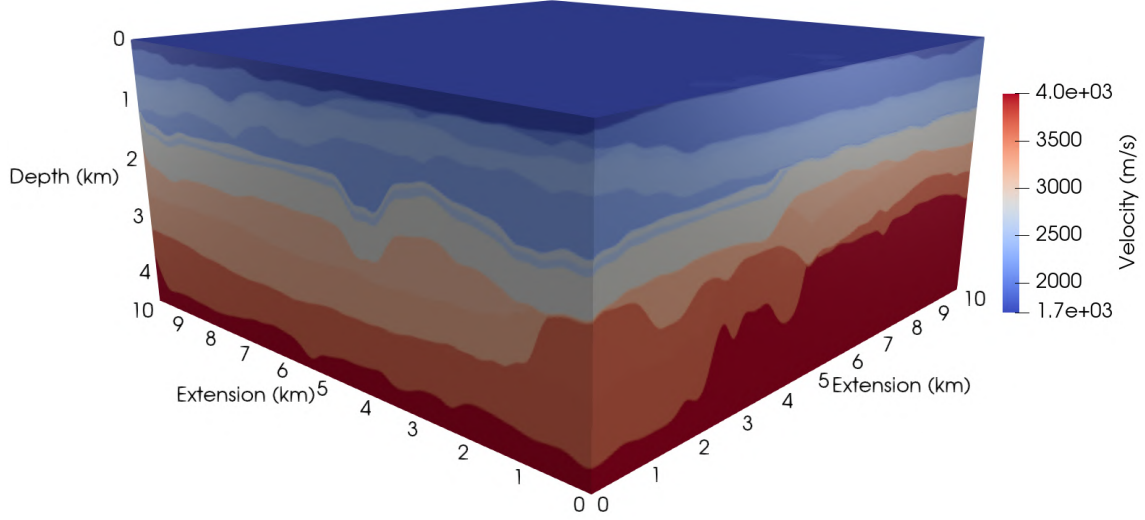


Figure 5.1: 3D velocity field provided by the HPC4E Seismic Test Suite.

### 5.1.1 Seismic modeling experiment

The seismic modeling experiment consists of propagating the wavefield on the HPC4E velocity model for a single shot located at  $[5000.0, 5000.0, 25.0]$  meters. As mentioned earlier, the total acquisition time is 6.0 seconds, which is the same for wavefield propagation. We have used two different grid spaces for the velocity field discretization in this numerical experiment, which are 25.0 meters and 12.5 meters. The grid spaces produce computational grids of  $501 \times 501 \times 235$  and  $901 \times 901 \times 416$  grid points, respectively. Both computational grids include  $N = 50$  for boundary conditions and half of the finite difference stencil length at the top of the velocity model to simulate the free surface. We did not implement none of the boundary conditions exposed in Appendix A because we intend to measure the computational performance only for the wave equation discretization.

We simulate the wavefield propagation based on Algorithm 1 for the vector processor optimizations and Algorithm 5 (and Listing 4.3) for the OpenACC implementation, both exposed in subsections 3.1 and 4.3. The seismic modeling implementation for multi-core machines follows the vectorization/OpenMP optimizations presented in subsection 4.1. Table 5.1 details the average time measurements of the seismic modeling on the NVIDIA V100 and SX-Aurora TSUBASA vector engine platforms. The second and fourth columns show the time measurements for the  $501 \times 501 \times 235$  grid size. The exe-

cution time is almost the same on both platforms, and it differs by approximately 1.0s. On the other hand, the wavefield simulation on the  $901 \times 901 \times 416$  grid size took 330.363 s for the NVIDIA V100 and 203.904 s for the SX-Aurora TSUBASA, representing an execution time difference of 126.423 seconds. The results show that both platforms have similar performances for the smallest grid. However, the SX-Aurora TSUBASA performs better on the largest grid. Notice that the 3D implementations do not show relevant system fluctuations for both platforms and grid sizes in the execution time measurements. The system fluctuations are represented by the variance columns.

Table 5.1: Seismic Modeling performance measurements for different grid sizes on the NVIDIA V100 and SX-Aurora TSUBASA. The average time is calculated for ten execution time measurements.

	NVIDIA V100		SX-Aurora TSUBASA	
	Av. Time (s)	Variance (s)	Av. Time (s)	Variance (s)
Grid 1: $501 \times 501 \times 235$	42.1505	0.0464	43.132	0.018
Grid 2: $901 \times 901 \times 416$	330.363	0.0464	203.940	0.177

Figures 5.2 and 5.3 show the seismic modeling speedup across the platforms Santos Dumont CPU Cluster, NVIDIA V100, and SX-Aurora TSUBASA vector processor for the  $501 \times 501 \times 235$  and  $901 \times 901 \times 416$  grids. The total speedup  $S_t$  consists of measuring improvements in execution time of an application executed on similar architectures with different resources [51, 56]. Thus,  $S_t$  can be formulated as follows:

$$S_t = \frac{T_s}{T_p}, \quad (5.1)$$

where  $T_s$  is the execution time of the application without parallelism (serial execution time) and  $T_p$  the execution time of the same application considering parallelism (parallel execution time). The serial execution time could be the time for the parallel implementation using one processor, the time for the corresponding serial implementation using one processor or the time for the "best" serial algorithm [56]. We have used the execution time measurement of the optimized serial implementation as the reference time to calculate the speedups. Therefore, the seismic modeling speedup for the optimized serial implementation is set as 1.0.

We run the optimized serial, OpenMP, and OpenACC implementations of seismic modeling on the Santos Dumont CPU-GPU Cluster. The optimized serial implementation runs on a single core, and the OpenMP version runs on 24 cores on a single node. Figure 5.2 shows that the OpenACC and vector processor implementations have practically the same performance for the  $501 \times 501 \times 235$  grid. The speedup of the seismic modeling for the OpenACC implementation on NVIDIA V100 is 36.22, and the speedup for the vector processor implementation on SX-Aurora TSUBASA is 35.4. Both speedups are

4.27 $\times$  faster than the OpenMP seismic modeling implementation. Remember that the SX-Aurora TSUBASA vector processor has 8 vector cores.

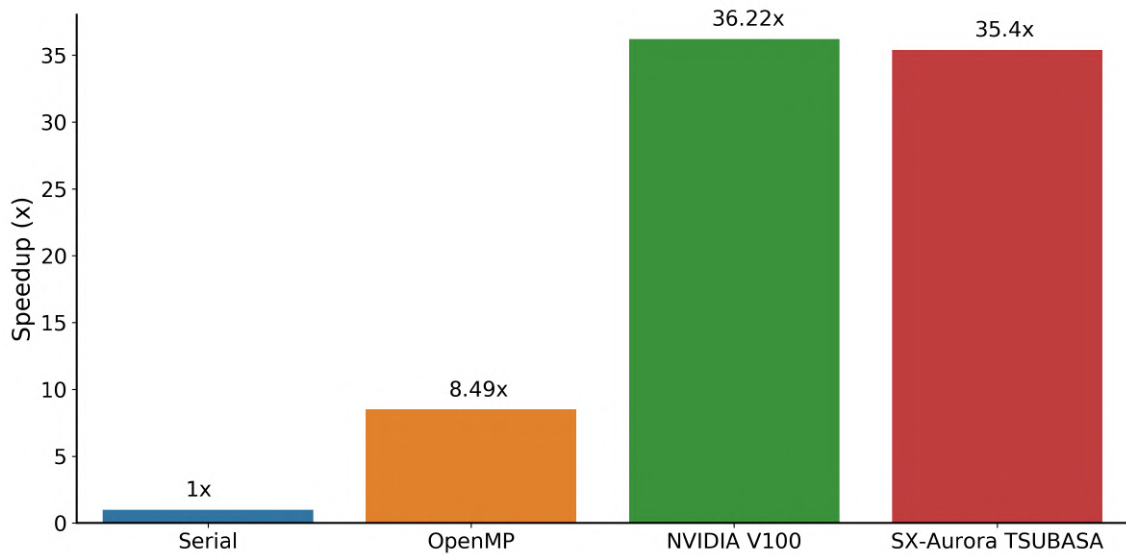


Figure 5.2: Seismic modeling speedup across the platforms Santos Dumont CPU Cluster, NVIDIA Volta V100 and SX-Aurora TSUBASA Vector Engine for the  $501 \times 501 \times 235$  grid.

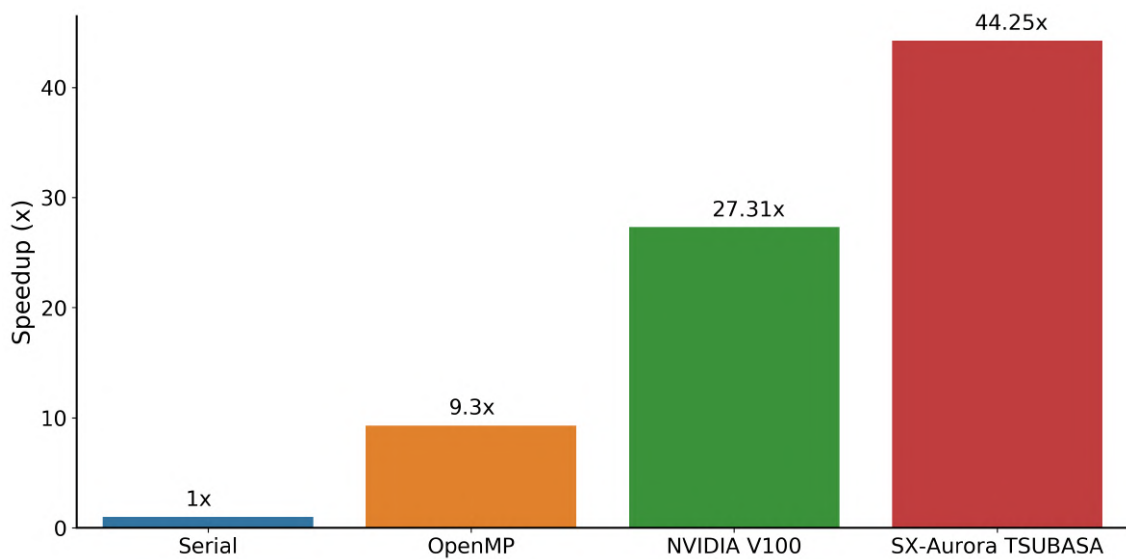


Figure 5.3: Seismic modeling speedup across the platforms Santos Dumont CPU Cluster, NVIDIA Volta V100 and SX-Aurora TSUBASA Vector Engine for the  $901 \times 901 \times 416$  grid.

The seismic modeling implementation on SX-Aurora TSUBASA has the best performance for the  $901 \times 901 \times 461$  grid size as shown in Figure 5.3. The speedup of the OpenMP implementation is 9.3, the OpenACC implementation speedup is 27.31, and the vector processor implementation speedup is 44.25. Thus, the OpenACC implementation on NVIDIA V100 performed 1.62 $\times$  worse than the vector processor implementation on

SX-Aurora TSUBASA and  $2.94\times$  better than the OpenMP implementation. The speedup of the vector processor implementation for the SX-Aurora TSUBASA is  $4.75\times$  better than the OpenMP implementation speedup.

We have used the `proginf`, `ftrace`, and `nvprof` tools to get information about the computational performance of the seismic modeling implementations. `proginf` and `ftrace` are profiling tools provided by NEC corporation, and `nvprof` is a profiling tool from NVIDIA. In this experiment, we propagate the wavefield during 0.6 seconds for the  $501 \times 501 \times 235$  grid. According to `proginf` and `ftrace` analysis, most of the time is spent on the wave equation calculation (`isotropicAcousticModeling()` function) corresponding to 99.1% of the total time. For the wave equation calculation, the vector operation ratio is 99.29%, which corresponds to 3.936 seconds with a cache miss of 0.008 seconds. The measurements show that our seismic modeling implementation presents a high vectorization ratio on the SX-Aurora TSUBASA vector processor (see Appendix G.1 for a detailed profiling report). The profiling results for the GPU implementation can be seen in Appendix G.2. Recall that it is essential to keep most of the calculation on the GPU aiming to minimize data transfer between the device and the host. Thus, according to the profiling analysis, 99.85% of the total time is spent on calculations on the GPU, and 0.15% of the total time corresponds to data transfer between the host and GPU device (see Figure 5.4). Our implementation is directed toward GPU devices that have high GPU usage and minimum data transferring for the seismic modeling implementation.

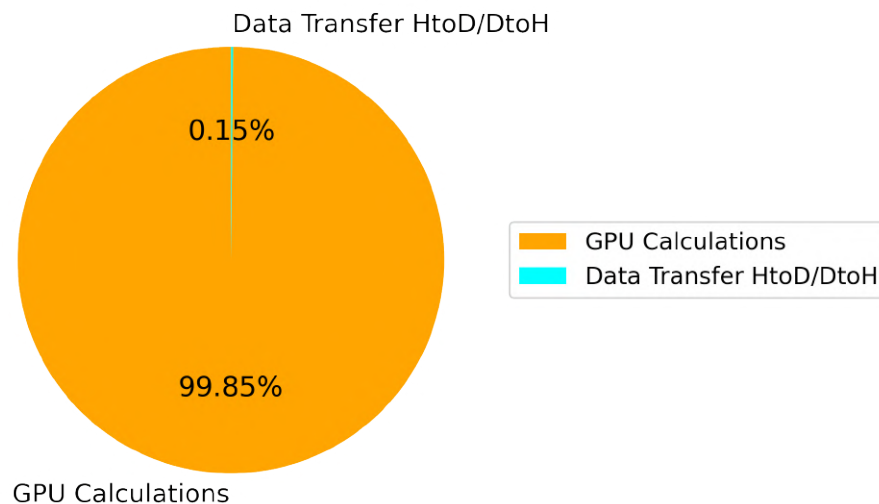


Figure 5.4: Percentage time spent on GPU calculations and data transfer for the seismic modeling.

Figure 5.5 shows three timeframes of the wavefield propagation in the 3D velocity field provided by the HPC4E Seismic Test Suite. The timeframes refer to the instants 0.6 s (upper), 1.5 s (middle), and 2.0 s (lower) for a single shot located at  $[x, y, z] = [5000.0, 5000.0, 25.0]$  meters. In this experiment, we run the computational

implementation of Algorithm 1 for the 3D case to generate the outputs shown in Figure 5.5. In this scenario, we implement an ABC based on CERJAN *et al.* [25] to avoid artificial reflections on the boundaries. The approach is different from the computational performance measurements only to present the wavefield propagation.

## 5.1.2 Seismic migration experiment

We presented in subsection 5.1.1 the seismic modeling computational performance for different platforms. The wave equation implemented for seismic modeling is the kernel of seismic imaging methods, such as the RTM. In the RTM process, the wave equation has to be solved two times, as implemented in Algorithm 2, and three times as implemented in Algorithm 4. However, the RTM computational cost is not proportional to the number of wave equations to be solved. The reasons are related to the incomes and outcomes of the RTM, which differs from the seismic modeling, and also related to how we deal with the source wavefield, an important partial result to build the imaging condition. Thus, we detail in this subsection the computational cost of RTM for single and multiple shot(s). We aim to understand which implementation performs better in terms of execution time and storage demand on different computational platforms.

**Single shot experiment** The first seismic migration experiment consists of executing the RTM application for a single seismic source (single shot) located at  $[5000.0, 5000.0, 25.0]$  meters. The RTM follows the implementations presented in Algorithms 2 and 4. We have used a  $501 \times 501 \times 235$  grid size with 25.0 meters of grid space to represent the velocity field. Notice that the grid size for the 3D cases includes  $N = 50$  for the RBC, and half of the finite difference stencil length at the top of the velocity model to simulate the free surface. The seismograms for the RTM are represented by the seismic signals recorded in a seismic survey. The receiver geometry of the seismogram follows the expressions:

$$r_x = 25.0(i - 1) + 1012.5 \text{ with } i = 1, \dots, 320, \quad (5.2)$$

$$r_y = 25.0(j - 1) + 1012.5 \text{ with } j = 1, \dots, 320, \quad (5.3)$$

where the pair  $[r_x, r_y]$  meters represents the receiver locations near the surface.

Table 5.2 shows the average time execution and the hard disk requirements for the 3D RTM implementations for the  $501 \times 501 \times 235$  grid. The RTM is based on Algorithms 2 and 6 implement the wavefield storage for the Nyquist time step described by equation (3.18), and Algorithms 4 and 7 describe the wavefield reconstruction. For the Ricker wavelet, which we use for the RTM test case,  $f_{max} = 100.0$  Hz and  $f_{min} = 0.0$  Hz. Thus, the Nyquist time step is  $\Delta t_{nyq} = 10.0$  ms and  $\Delta t = 1.0$  ms, 10 times bigger than the FDM time step. Hence, Algorithms 2 and 6 which implement the wavefield storage require

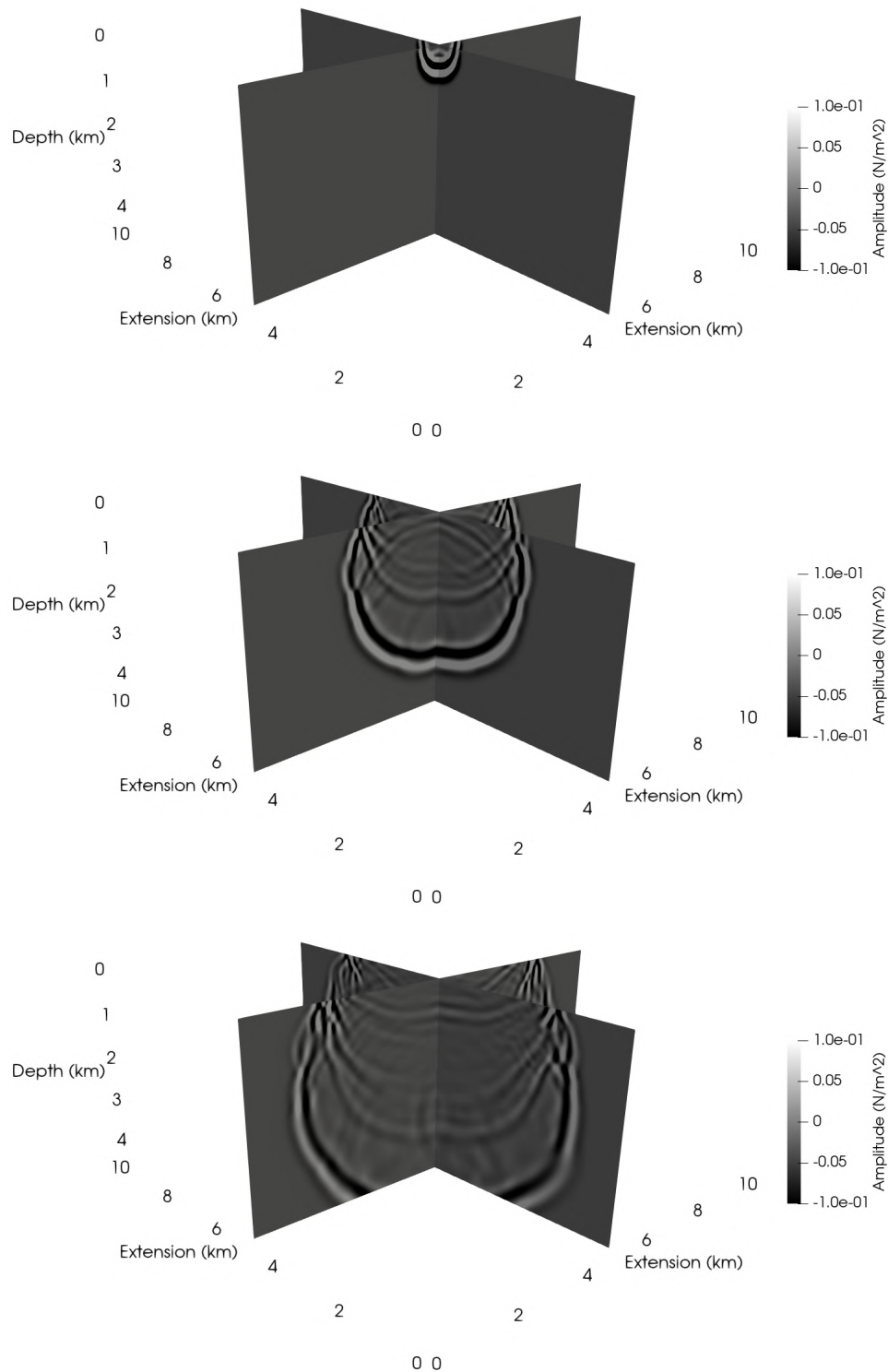


Figure 5.5: Propagation of the wavefield in the 3D velocity field provided by the HPC4E Seismic Test Suite for the instants 0.6 s (upper), 1.5 s (middle), 2.0 s (lower). The results were provided by Algorithm 1.

132.062 GB of hard disk on NVIDIA V100 and SX-Aurora TSUBASA against 0.439 GB of the wavefield reconstruction implementations (Algorithms 4 and 7). This represents  $300.82\times$  less information to be stored.

The best average execution times refer to the RTM that implements the wavefield reconstruction on NVIDIA V100 and SX-Aurora TSUBASA. Nevertheless, the vector processor implementation of the RTM performed better than the OpenACC implementation, which is 108.582 s for the vector processor implementation against 139.786 s for the OpenACC implementation. Considering the wavefield storage implementations in Algorithms 2 and 6, the best average execution time is from NVIDIA V100, where the OpenACC implementation takes 256.166 s to run, and the vector processor 430.944s. The RTM with wavefield reconstruction is  $1.83\times$  faster than the wavefield storage implementation for the NVIDIA V100 and  $3.08\times$  faster than the wavefield storage implementation for the SX-Aurora TSUBASA. The average execution time for the RTM implementation with wavefield reconstruction on SX-Aurora TSUBASA is  $3.98\times$  better than the wavefield storage on the same platform,  $1.29\times$  and  $2.36$  better than the wavefield reconstruction and wavefield storage implementations for the NVIDIA V100.

Table 5.2: Comparison of hard disk and time requirements for the 3D RTM implementation with the wavefield storage, and the wavefield reconstruction.

Method	Platform	Hard disk (GB)	Av. Time (s)[Variance(s)]
Storage	NVIDIA V100	132.062	256.166 [46.810]
Reconstruction	NVIDIA V100	0.439	139.786 [0.293]
Storage	SX-Aurora TSUBASA	132.062	430.944 [5.716]
Reconstruction	SX-Aurora TSUBASA	0.439	108.582 [0.049]

Comparing the RTM speedups across the platforms Santos Dumont CPU Cluster, NVIDIA V100, and SX-Aurora TSUBASA, we can see in Figure 5.6 that the OpenMP implementation speedup is 12.09, the OpenACC implementation speedup is 54.62, and the vector processor implementation speedup is 70.32. All the implementations are based on Algorithms 4 and 7 which describe the RTM with the wavefield reconstruction. The RTM vector processor implementation has the best performance, and it is  $5.82\times$  better than the OpenMP implementation and  $1.28\times$  better than the OpenACC implementation. Lastly, the performance of the RTM implementation with OpenACC is  $4.52\times$  OpenMP implementation.

Again, we have used the `proginf`, `ftrace`, and `nvprof` tools, but in this case, the information about the computational performance is related to the seismic migration implementations. Note that the RTM that implements the wavefield reconstruction solves an extra wave equation leading to a total of three of them. Analyzing the execution time for RTM with wavefield reconstruction, we can conclude that such values are more or less three times the execution time of the seismic modeling for both NVIDIA V100 and

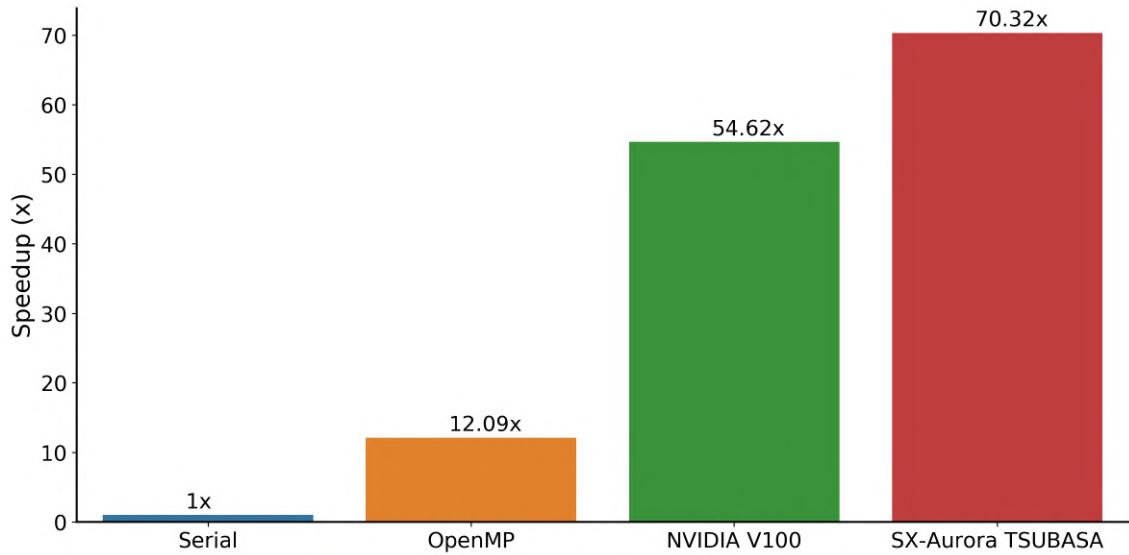


Figure 5.6: Reverse Time Migration speedup across the platforms Santos Dumont CPU Cluster, NVIDIA V100 and SX-Aurora TSUBASA Vector Engine for the  $501 \times 501 \times 235$  grid.

SX-AURORA TSUBASA. Thus, the extra time spent on the RTM that implements the wavefield storage is related to data transfer. Such implementations are bounded by data transfer between the host and the device. Because of that, we choose to get the profiling analysis only for the RTM with wavefield reconstruction to show that this implementation can minimize the data transfer and maximize the device usage. According to `proginf` and `ftrace` profiling reports, 87.3% of the total time corresponds to wave equation calculations (`iso_isotropic_wave_equation()` function), 7.6% to generate the random velocities for the RBC (`random_damping_layers_carrilho()` function), and 4.2% to build the seismic image (`cross_correlation()` function). Profiling analysis on the VE yield performance measurements separated by functions as we can see in Figure 5.7. The vector operation ratio is 99.3% for the wave equation calculation, 0.0% for the random velocity calculations, and 98.9% for the seismic image calculation. For the vectorized functions, the cache miss value is 0.18%. The profiling results indicate a high vectorization ratio. The profiling report for the NVIDIA GPU shows that only 0.07% of the total time is spent on data transfer between the host and GPU showing that the RTM with wavefield reconstruction minimizes data exchange (see Figure 5.8). Further details about the computational performance on SX-Aurora TSUBASA and NVIDIA GPU can be seen on Appendix G.

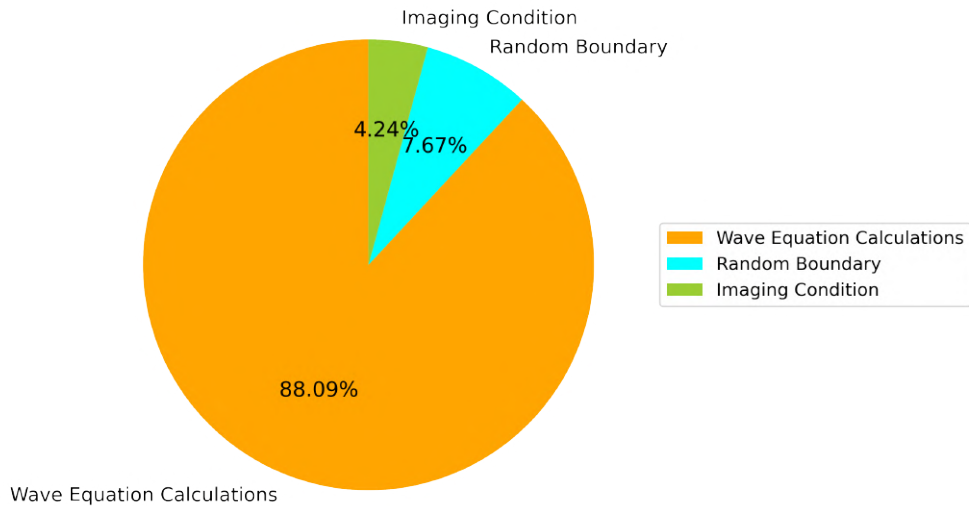


Figure 5.7: Percentage time spent on the VE Aurora TSUBASA to generate the random boundary, solve the wave equation, and calculate the convolutional imaging condition for the RTM based on the source wavefield reconstruction.

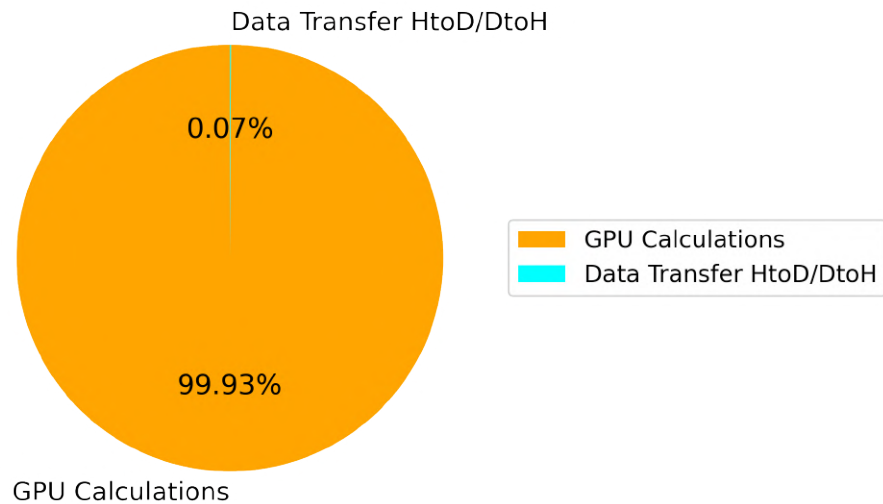


Figure 5.8: Percentage time spent on GPU calculations and data transfer for the RTM based on the source wavefield reconstruction.

Figure 5.9 shows the partial seismic image for one migrated shot of the 3D HPC4E Seismic Test Suite benchmark. The shot is located at  $[x, y, z] = [5000.0, 5000.0, 25.0]$  meters with the Ricker wavelet signature of a cutoff frequency of 20.0 Hz. We generated the observed seismogram by simulating the wave propagation and recording the seismic signals at the locations following the equations (5.2) and (5.3) near the surface at 25.0 meters in depth.

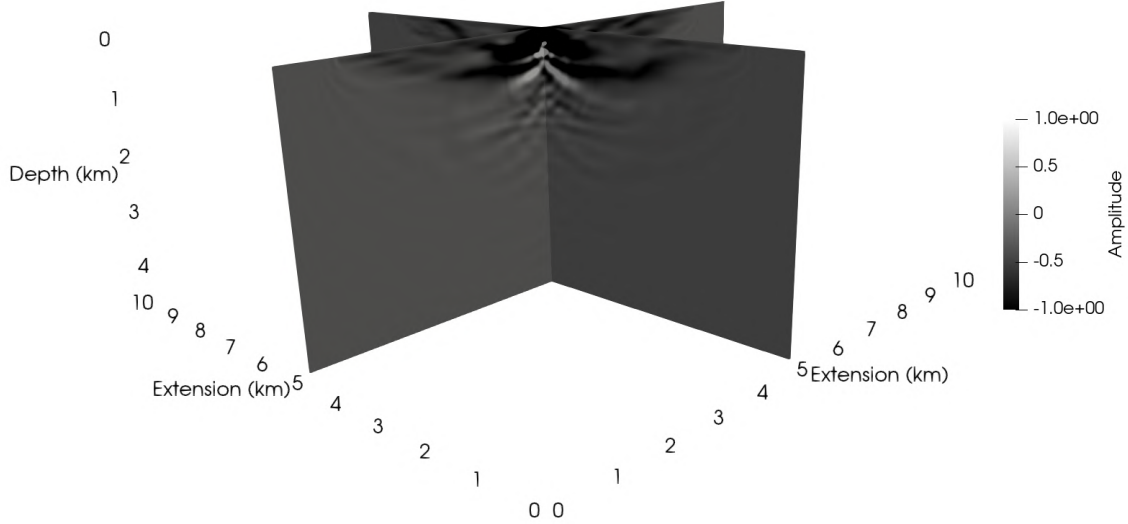


Figure 5.9: Seismic image for one migrated shot of 3D HPC4E Seismic Test Suite.

**Seismic survey experiment** The second experiment considers running the RTM for a survey geometry with 1681 seismograms. The geometry acquisition for the seismic survey follows the expressions:

$$s_x = 200.0(i - 1) + 1000.0 \text{ with } i = 1, \dots, 41, \quad (5.4)$$

$$s_y = 200.0(j - 1) + 1000.0 \text{ with } j = 1, \dots, 41, \quad (5.5)$$

where the pair  $[s_x, s_y]$  meters represents the seismic source locations near the surface.

Firstly, we compare the total time of executing the RTM on the CPU Cluster and NVIDIA V100. For the CPU cluster, we consider 16 nodes, where each node has 24 physical cores. Considering the GPU machine, we set 4 nodes because each one has 4 NVIDIA V100 adding up to 16 GPUs. Table 5.3 shows the total time of executing the RTM with wavefield reconstruction on the CPU cluster and NVIDIA V100. We can see that the OpenACC implementation performs better than the OpenMP implementation for the same strategy, which eliminates I/O related to the source wavefield. In this experiment, the RTM considering 16 NVIDIA GPUs is 7.62 times faster than the RTM running on 16 CPUs nodes. Besides, the RTMs based on the wavefield reconstruction demanded the same space in the disk, which is 7.024 GB.

Table 5.3: Comparison of the total execution time for the 3D RTM implementation with wavefield reconstruction on CPU cluster and NVIDIA V100.

Method	Platform	Total Time
Wavefield Reconstruction	CPU Cluster	1259min 22.560s
Wavefield Reconstruction	NVIDIA V100	170min 56.420s

Figure 5.10 shows the stacked seismic image for the 3D HPC4E Seismic Test Suite

benchmark. We generated the observed seismograms by simulating the wave propagation and recording the seismic signals at the locations following the equations 5.2 and 5.3 near the surface at 25.0 meters in depth. The survey acquisition follows the geometry expressed in equations 5.4 and 5.5 and takes into account 1681 shots.

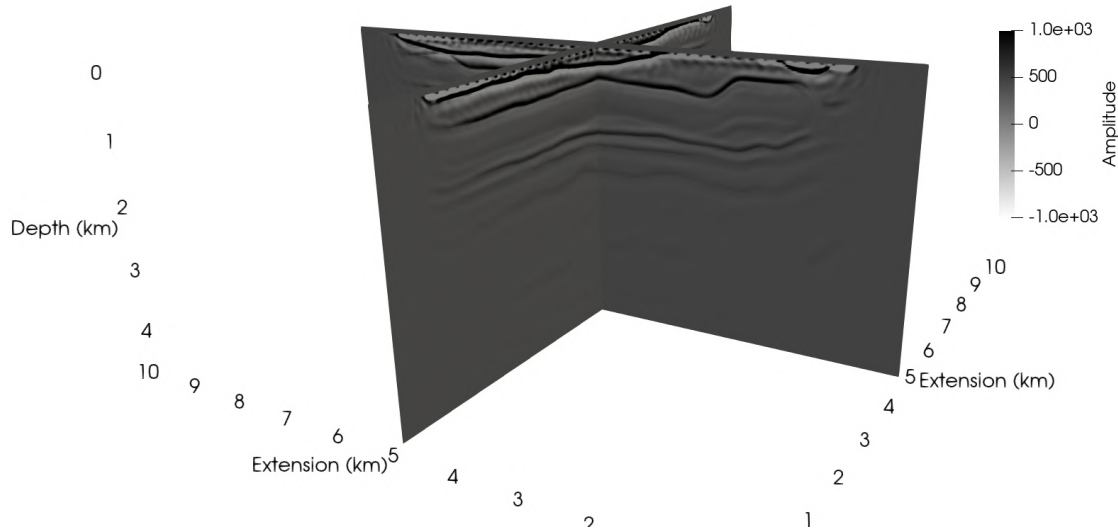


Figure 5.10: Seismic image for the 3D HPC4E Seismic Test Suite.

## 5.2 Data compression

There are two critical stages concerning the storage demand for RTM applications, and both are equally valid for techniques based on two-way wave equations. The first one is related to the number of seismograms to be stored for a seismic imaging application. This number can reach hundreds of thousands of seismograms depending on the area to be covered. And of course, the number of seismograms, even decimated, impacts directly the storage demand. Another crucial stage concerning the RTM technique is related to the requirement of storing the source wavefield in a disk to further build the imaging condition. In this case, the disk requirement depends on the acquisition domain, time step, and the number of applications running concurrently. Aiming to mitigate the storage demand for both scenarios, we describe in this section the impact of compressing the seismograms and/or source wavefield for the RTM technique. The impact is analyzed in terms of storage demand, execution time, and also numerical differences among the solutions. We use the 2D Marmousi and Marmousi2 benchmarks to analyze the numerical differences among the RTM solutions. After that, we present the storage demand, execution time, and overhead for a 3D scenario based on the HPC4E benchmark. The computational measurements are made on a CPU (Intel Xeon E5-2695v2 Ivy Bridge) and a GPU (Intel Xeon Skylake 6252) platforms, both from the Santos Dumont cluster<sup>4</sup> already described

<sup>4</sup>[https://sdumont.lncc.br/support\\_manual.php?pg=support](https://sdumont.lncc.br/support_manual.php?pg=support)

in section 5.1.

### 5.2.1 Seismogram compression

The first experiment in this section focus on seismogram compression and its effect on the final seismic image. We select the 2D Marmousi2 velocity model benchmark [121] (Figure 5.11). The Marmousi2 benchmark is 3.5 km depth and 17.0 km in the horizontal direction and has parameter models (p-wave velocity and density), and other data sets related to its acquisition. However, we choose to synthesize the observed seismograms solving the two-way wave equation with the reference velocity field provided by the benchmark. Therefore, we simulate a fixed split-spread acquisition [62] comprising 160 shots, where the seismic source, with a cutoff frequency of 45 Hz, is placed near the surface and moved 100 meters for each shot. Near-surface hydrophones record the seismic signals that compose the seismograms, and the receivers are equally spaced of 25 meters. The computational grid is  $2721 \times 561$ . Thus, we have 160 seismograms to test the lossy and lossless compression rates and to measure the error propagation due to seismogram compression on the final seismic images.

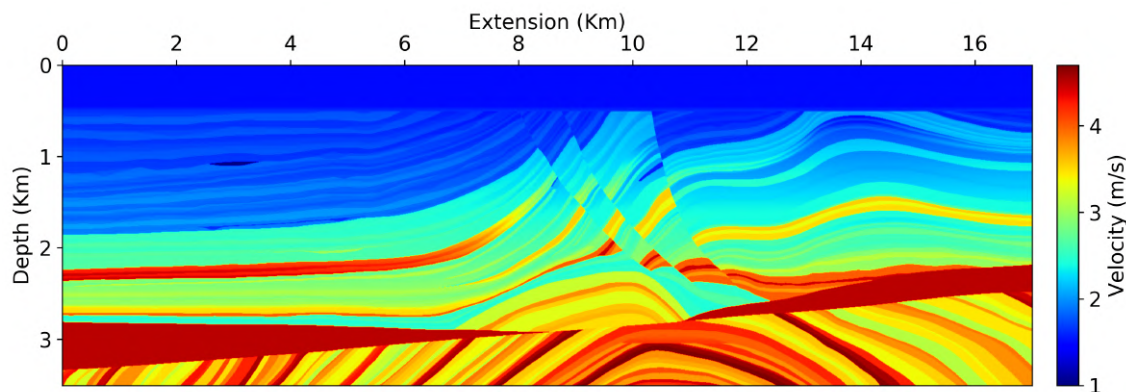


Figure 5.11: 2D velocity field from the Marmousi2 benchmark.

We begin the data compression analysis by applying lossless and lossy compression to the 160 seismograms. The assumed absolute error for the lossy compression is  $10^{-4}$ . Following this, we run the RTM without compression to get reference outcomes for the error measurements related to the results of compressed seismograms. To get the seismic image reference results, we implement the RTM based on Algorithm 2. Figure 5.12 (upper) shows the migration of the uncompressed seismograms, and Figure 5.12 (lower left and right) show the 2D Fourier spectra of the two white boxes highlighted in Figure 5.12 (upper).

In a second moment, we run RTM using the lossless and lossy (tolerance error of  $10^{-4}$ ) seismogram compressions as inputs for RTM migration. For this, we adjust Algorithm 2 to read the compressed seismogram by changing its line 14, which reads the

seismograms without compression, for lines 6 and 7 of Algorithm 8. Thus, line 6 reads the compressed seismogram and line 7 decompresses the seismogram for each shot iteration of the RTM.

---

**Algorithm 8** Reverse Time Migration with Compressed Seismograms

---

**Require:**  $\mathbf{v}$ ,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , and  $\mathbf{f}$

- 1: **function** RTM( vector  $\mathbf{v}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
- 2:     read  $\mathbf{v}$ , and  $\mathbf{f}$
- 3:     initialize image condition  $\mathbf{I}_{\Sigma shot\_id} = 0$
- 4:     **for**  $shot\_id = 1$  to  $N_{shots}$  **do**
- 5:         source wavefield calculation...
- 6:         read compressed seimogram  $\mathbf{s}_{shot\_id}$
- 7:         decompress seimogram  $\mathbf{s}_{shot\_id}$
- 8:         receiver wavefield calculation...
- 9:         stacking...
- 10:     **end for**
- 11:     storing seismic image
- 12: **end function**

---

The results based on Algorithm 8 can be seen in Figures 5.13 and 5.14, respectively. No visual difference can be identified among the migrated seismic sections and 2D Fourier spectra, but we compute the normalized root mean square (NRMS) error among the RTM results from Figure 5.12 (upper), and the RTM results from Figures 5.13 (upper), and 5.14 (upper) to get a numerical representation of the errors. The NRMS errors are  $2.0 \times 10^{-6}\%$  and  $4.3 \times 10^{-5}\%$ , respectively. The NRMS errors among the spectra from the left white box regions in Figures 5.12 (upper), 5.13 (upper) and 5.14 (upper) are  $1.1 \times 10^{-6}\%$ , and  $1.97 \times 10^{-5}\%$ , respectively. Moreover, the NRMS errors among the spectra from the right white box regions in Figures 5.12 (upper), 5.13 (upper) and 5.14 (upper) are  $1.3 \times 10^{-6}\%$ , and  $2.6 \times 10^{-5}\%$ , respectively. The results from Figures 5.12, 5.13, 5.14 and the NRMS errors show that the chosen compression levels for the seismograms lead to migrated images with errors of order  $10^{-5}\%$ .

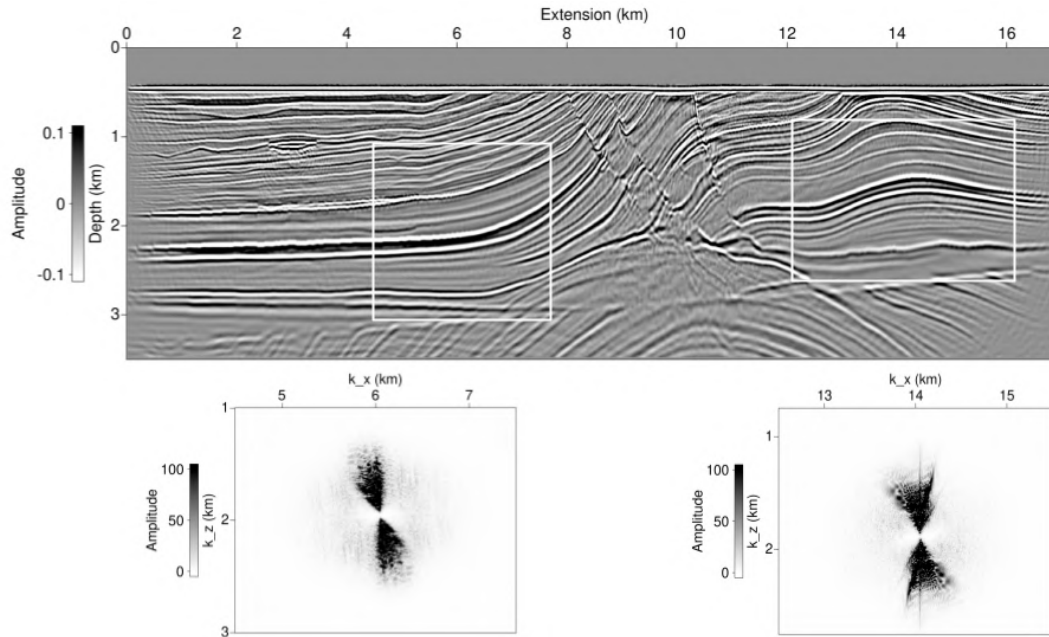


Figure 5.12: RTM outcome after migrating the reference seismograms (upper image). Figures lower left and lower right represent the 2D Fourier spectra of the white box regions shown in the migrated seismic image.

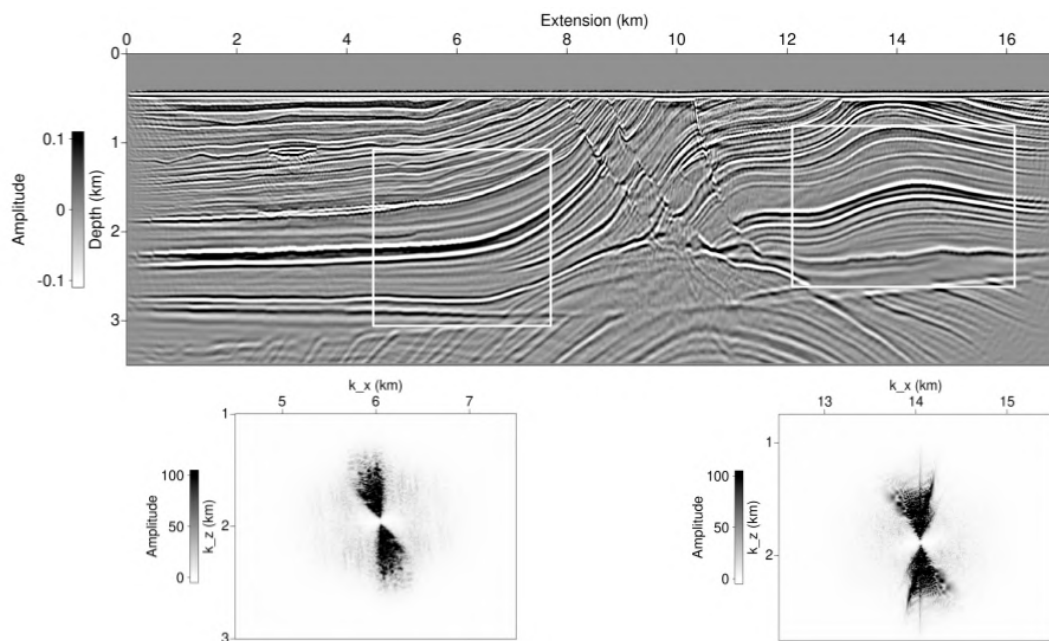


Figure 5.13: RTM outcome after migrating the lossless compressed seismograms (upper image). Figures lower left and lower right represent the 2D Fourier spectra of the white box regions shown in the migrated seismic image.

The results from Figures 5.12, 5.13, and 5.14 suggest that we can use compressed data on the seismic imaging process without hampering the migrated images and, thus, reduce their storage and network data transfer. Aiming to exemplify the storage reduction, Table 5.4 shows that each seismogram of our numerical test has 76.20 MB, and we can

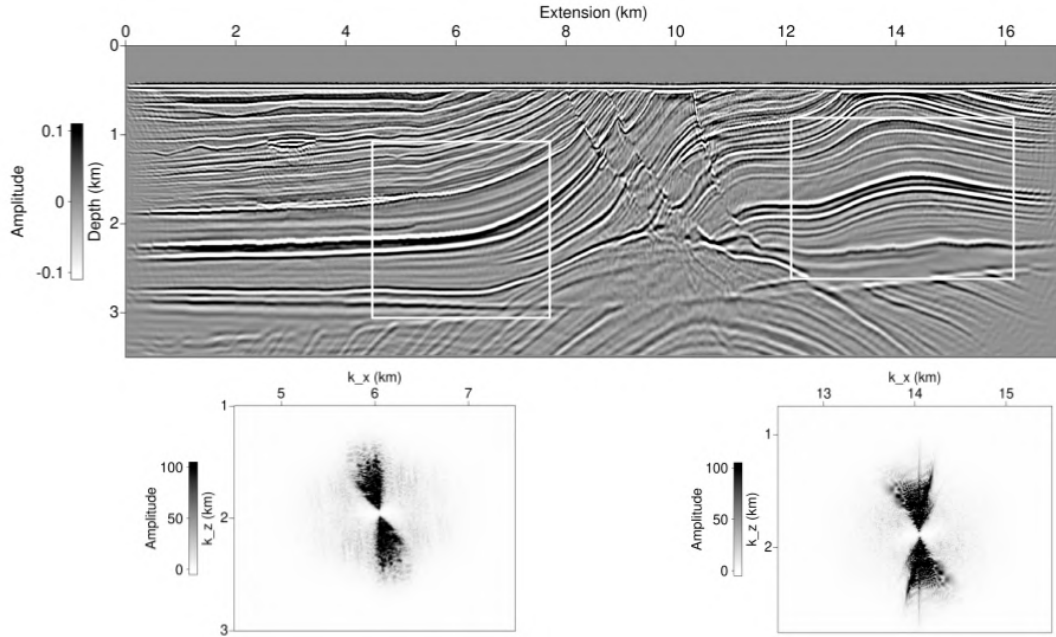


Figure 5.14: RTM outcome after migrating the lossy compressed seismograms (upper image). Figures lower left and lower right represent the 2D Fourier spectra of the white box regions shown in the migrated seismic image.

conclude that the demand of total storage is 12.0 GB for the 160 seismograms. Storage demand after applying the lossless and lossy compression can also be seen in Table 5.4. These values represent 82.4%, and 89.0% less than the original data, respectively.

Table 5.4: Seismograms fixed accuracy lossy and lossless compression comparison. The error tolerance for lossy compression is set to  $10^{-4}$ . The values represent the minimum and maximum compression among the 160 seismograms.

Compression Type	Minimum (MB)	Maximum (MB)
None	76.20	76.20
Lossless	59.28	48.46
Lossy	13.38	8.38

**Storage demand for 3D environments** We chose a 2D scenario based on the Marmousi2 benchmark to measure the error propagation due to compression on the final seismic image building process. The results shown in Figures 5.12, 5.13, and 5.14 suggest that we can use compressed data on the seismic imaging process without hampering the migrated images. We discuss in this section only the storage demand and the overhead to decompress the 1681 seismograms used for the RTM technique based on the 3D HPC4E benchmark. The 3D RTM follows the same configurations as presented in section 5.1.2, which are the velocity field shown in Figure 5.1, seismic source of the cutoff frequency

of 20.0Hz, and shot and receiver geometries following Equations 5.2, 5.3, 5.4, and 5.5. We use the RTM implementations based on Algorithms 2 and 8 to measure the storage demand and overhead values for the chosen 3D scenario.

Table 5.5 shows that each seismogram for the 3D RTM numerical test has 469.14 MB without applying data compression. Thus, the total storage demand on disk is 770.14 GB for the 1681 seismograms. The storage demand after applying the lossless and lossy compression can also be seen in Table 5.5, where the error tolerance for the lossy compression is  $10^{-4}$ . The values represent the minimum and maximum compression among the 1681 available seismograms, and they represent approximately 14.0%, and 54.5% less information to be stored than the original data set for each seismogram, respectively.

Table 5.5: Seismograms fixed accuracy lossy and lossless compression comparison. The error tolerance for lossy compression is set to  $10^{-4}$ . The values represent the minimum and maximum compression among the 1681 seismograms for the 3D HPC4E benchmark.

Compression Type	Minimum (MB)	Maximum (MB)
None	469.14	469.14
Lossless	402.98	398.66
Lossy	213.64	201.07

Considering the RTM application, where it is necessary to read the compressed seismograms, the overhead values to decompress the seismograms depend on the platform where the RTM computational implementation is run, and also the decompression rate (lossless or lossy decompression). Unfortunately, it was not possible to get computational measurements for the 3D RTM which implements the data compression on the VE (SX-Aurora TSUBASA) due to its unavailability when the compression tests were done. Thus, we run the 3D RTM implementations on the CPU (Intel Xeon Ivy Bridge) and GPU (NVIDIA V100) platforms used for the 3D seismic migration experiments to measure the overhead values. Table 5.6 shows that the overhead values are 5.282s and 3.305.4s using the lossless and lossy decompression on the CPU node, and 4.149s and 2.604s using the lossless and lossy decompression on the GPU node.

The overheads of decompressing the seismograms on the CPU platform correspond to 0.83% and 0.53% of the total execution times for the lossless and lossy compressions. For the GPU platform, the decompression overheads correspond to 2.87% and 1.83% of the total execution times, also for the lossless and lossy compressions, respectively. The measurements suggest that our RTM linked to the ZFP compression can provide accurate seismic images (as we saw in Figures 5.12, 5.13, and 5.14) with negligible NRMS errors (order of  $10^{-5}$ ) with overhead less than  $\approx 3\%$ . The RTMs that implement the compression in both CPU and GPU platforms generate small overheads and do not compromise the

Table 5.6: Time requirements and overhead for the 3D RTM implementation considering different compression rates for the seismograms. The error tolerance for lossy compression is set to  $10^{-4}$ . The measurements take into account ten executions of each implementation.

Platform	Method	Time (s)[Var. (s)]	Overhead (s)[Var. (s)]
CPU	None	631.474 [0.112]	-
CPU	Lossless	635.854 [0.249]	5.282 [0.004]
CPU	Lossy $10^{-4}$	633.116 [0.152]	3.305 [0.002]
NVIDIA V100	None	139.786 [0.293]	-
NVIDIA V100	Lossless	144.805 [0.391]	4.149 [0.050]
NVIDIA V100	Lossy $10^{-4}$	142.571 [0.286]	2.604 [0.052]

total execution time.

## 5.2.2 Wavefield compression

Two critical stages exist when dealing with storage demands for RTM. The first is related to the number of seismograms as discussed in section 5.2.1, and the second is related to temporary information, such as the source wavefield, while RTM is running. Here, we deal with the need of storing the source wavefield, and we have chosen data compression for that. The data compression analysis is done by applying lossless and lossy compression to the source wavefield using the ZFP library linked to our RTM implementation (Algorithm 3). The assumed absolute errors for the lossy compression are  $10^{-4}$  and  $10^{-6}$ .

To illustrate RTM results considering the source wavefield compression, we choose the 2D Marmousi benchmark, which is 3.0 km in depth and 9.2 km in the horizontal direction (Figure 5.15). We choose to synthesize the observed seismograms solving the two-way wave equation with the reference velocity field provided by the benchmark. Therefore, we simulate a fixed split-spread acquisition [62], where the seismic source, with a cutoff frequency of 45 Hz, is placed on the surface and moved 100 meters for each shot. Near-surface hydrophones record the seismic signals that compose the seismograms, and the receivers are equally spaced of 12.5 meters. Thus, the dataset for seismic migration is composed of 82 seismograms.

Figure 5.16(a) shows the resulting seismic image without using compression that we define as the reference outcome. Figure 5.16(b), (c), and (d) show the migrated seismic images built by RTM linked to the compression library. These results apply the lossless compression, the lossy tolerance compression of  $10^{-6}$ , and the lossy tolerance compression of  $10^{-4}$  to source wavefields. Observe that the aggressive lossy tolerance compression of  $10^{-4}$  damages the final seismic image (Figure 5.16(d)). However, we do not note any damage on the seismic images in Figures 5.16(b), and (c).

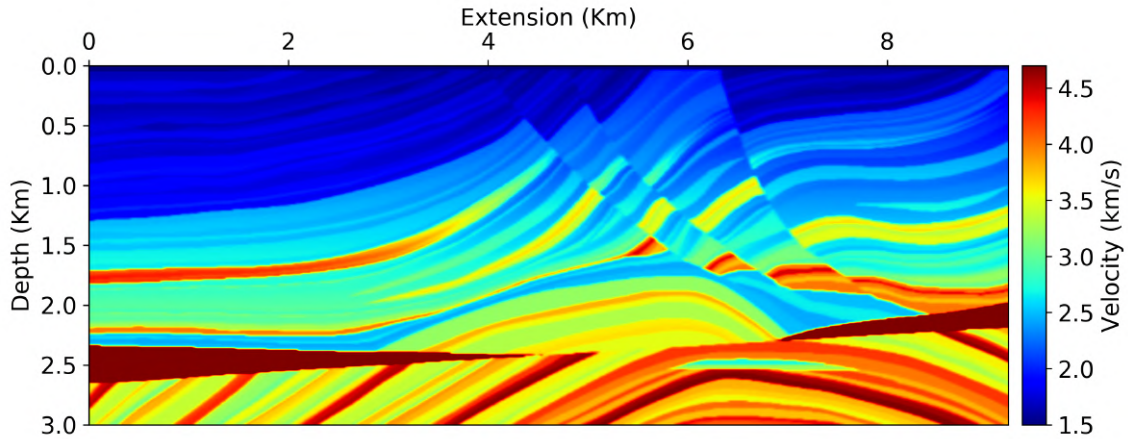


Figure 5.15: 2D velocity field from the Marmousi benchmark.

The errors produced by the difference between the reference image and seismic images built using compression can be seen in Figure 5.17. Figure 5.17(a) shows that lossless compression of the source wavefield provides the best result. In this case, the error is of order  $10^{-6}$ . On the other hand, a lossy tolerance compression of  $10^{-4}$  of the source wavefield produces the most inferior outcome. Lastly, a lossy tolerance compression of  $10^{-6}$  suggests that lossy compression can also be an option to deal with the storage of source wavefields. In this case, the error is of order  $10^{-3}$ .

The total demand to store the source wavefield in disk without compression is 1.8 GB per shot, and after applying compression is 1.1 GB for lossless compression, 92.9 MB for the lossy tolerance compression of  $10^{-6}$ , and 15.2 MB for the lossy tolerance compression of  $10^{-4}$ . These values represent 38.9%, 94.8%, and 99.2% less than the original data. Figure 5.18 shows the storage demand per time slice for the source wavefield without compression (blue line), with lossless compression (orange line), with a lossy tolerance compression of  $10^{-6}$  (red line), and with a lossy tolerance compression of  $10^{-4}$  (green line). We see that the size varies for each time step when we use data compression. This behavior occurs because amplitude information changes during wavefield propagation.

**Storage demand for 3D environments** The 2D RTM results shown in Figures 5.16 and 5.17 also suggest that compression can be a good strategy to reduce the storage demand concerning the source wavefield. We showed for a 2D scenario that the storage demand reduced 38.9% and 94.8% for the lossless compression, and lossy tolerance compression of  $10^{-6}$  with errors of order of  $10^{-6}$  and  $10^{-3}$ , respectively. Following, we discuss in this section the storage demand and overhead of storing the source wavefield for the 3D HPC4E benchmark. Again, the 3D RTM follows the same configurations as presented in section 5.1.2, which are the velocity field shown in Figure 5.1, seismic source of the cutoff frequency of 20.0Hz, and shot and receiver geometries following Equations 5.2, 5.3, 5.4, and 5.5. We used the RTM implementations based on Algorithms 2 and 3 to measure the

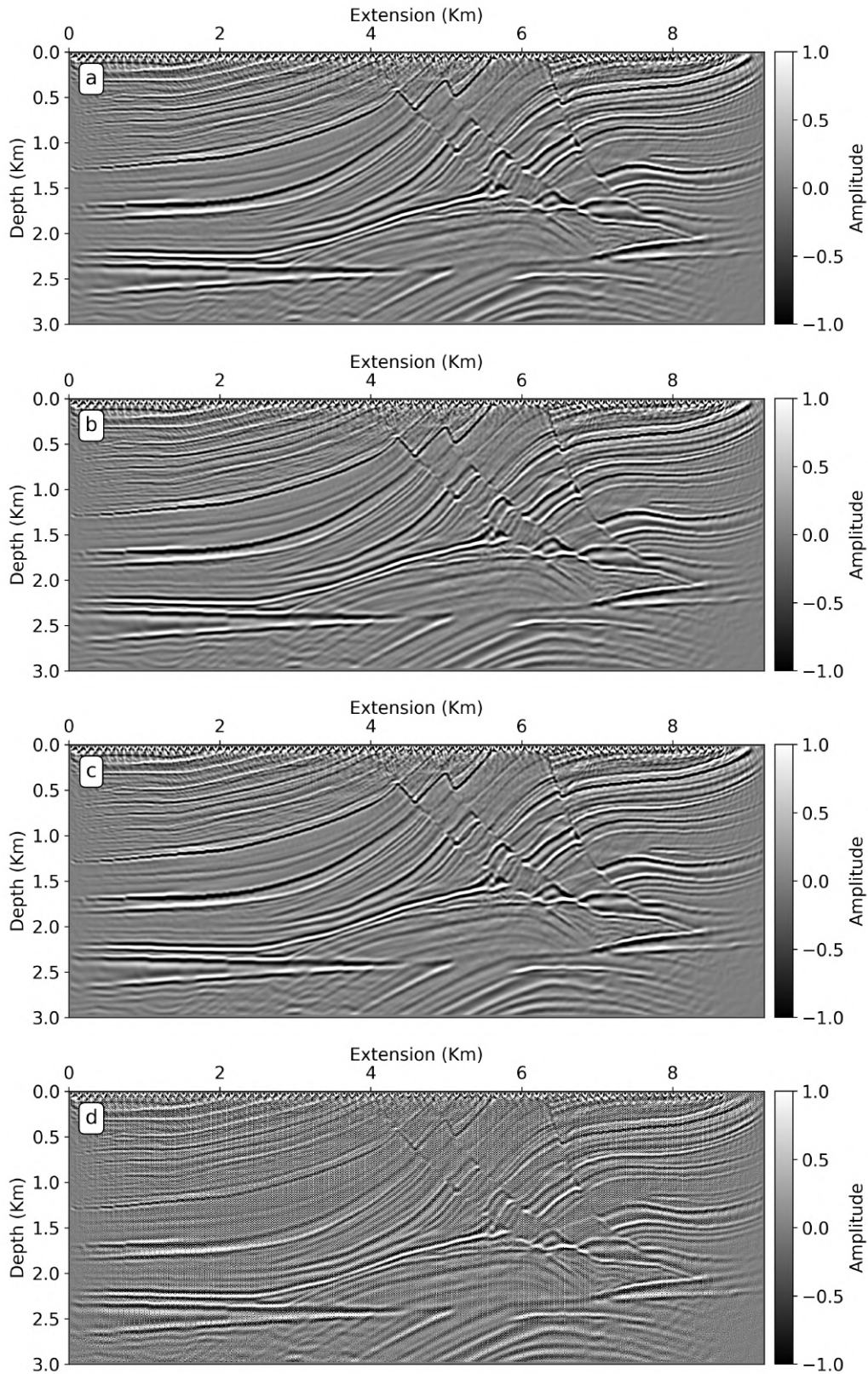


Figure 5.16: RTM outcomes after migration using the forward-propagated source wave-field (a) without compression, (b) with lossless compression, (c) with a lossy tolerance compression of  $10^{-6}$ , and (d) with a lossy tolerance compression of  $10^{-4}$ .

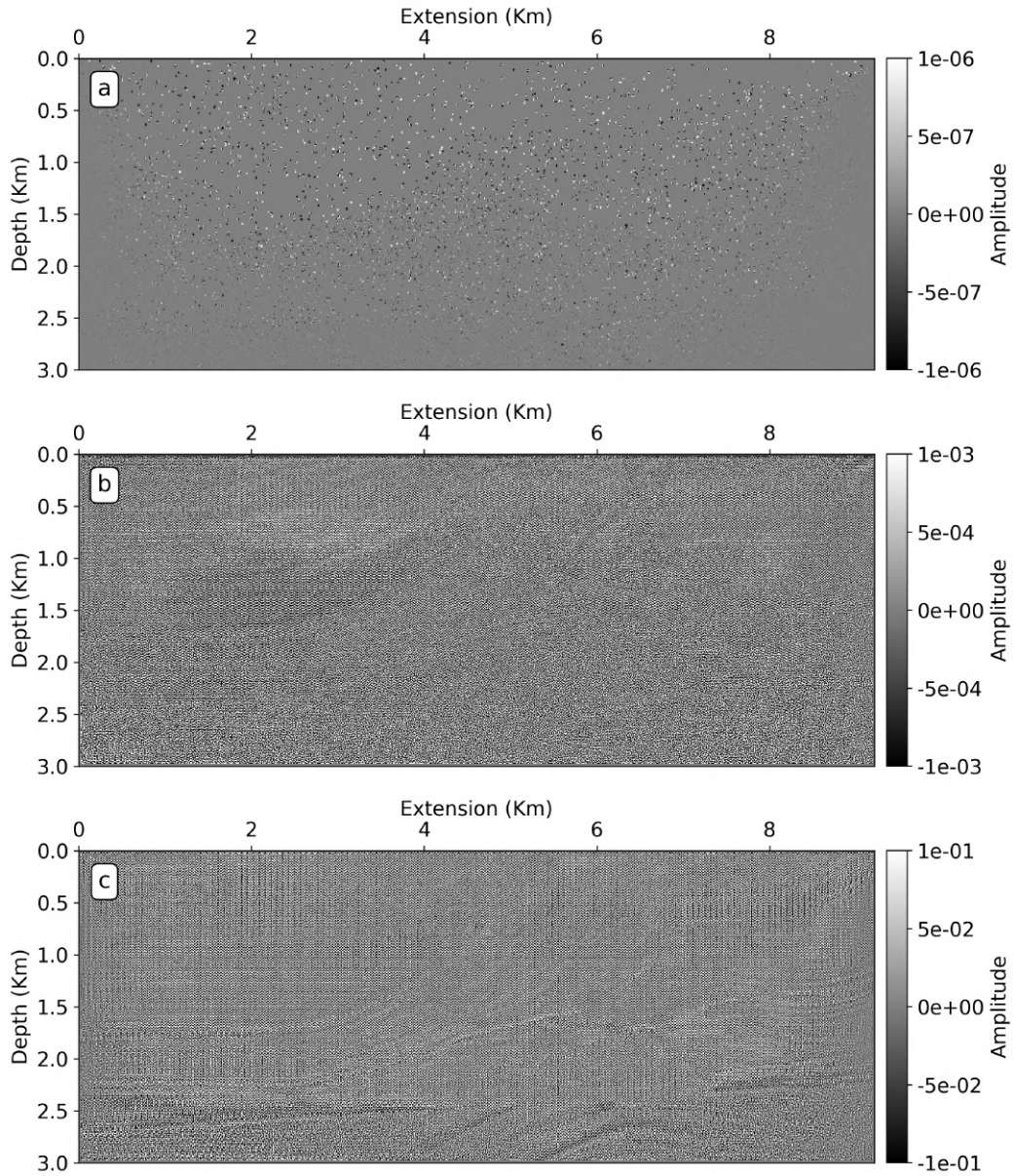


Figure 5.17: Difference among the reference migrated image (no compression) and (a) the migrated image after applying lossless compression on the source wavefield, (b) the migrated image after applying a lossy tolerance compression of  $10^{-6}$  on the source wavefield, and the migrated image after applying a lossy tolerance compression of  $10^{-4}$  on the source wavefield.

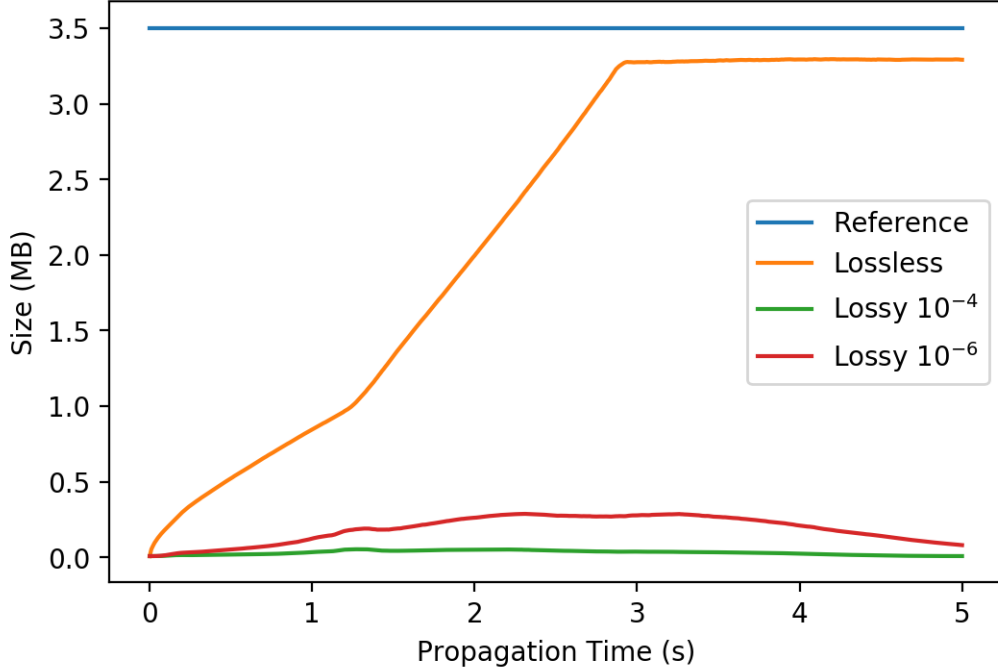


Figure 5.18: Size per time slice for the 2D source wavefield without compression (blue line), with lossless compression (orange line), with a lossy tolerance compression of  $10^{-4}$  (green line), and with a lossy tolerance compression of  $10^{-6}$  (red line).

storage demand and overhead values for the chosen 3D scenario.

Figure 5.19 shows the storage demand in disk per time slice for the source wavefield without compression (blue line), with lossless compression (orange line), with a lossy tolerance compression of  $10^{-6}$  (green line), and with a lossy tolerance compression of  $10^{-4}$  (red line). Note that size changes for each time step when we apply the data compression similar to the 2D experiment from Figure 5.18. Again, this behavior occurs because amplitude information changes during the wavefield propagation. On the other hand, the total demand to store the source wavefield in disk for all time steps without compression is 65.16 GB per shot, and after applying compression is 49.82 GB for lossless compression, 31.34 GB for the lossy tolerance compression of  $10^{-6}$ , and 21.68 GB for the lossy tolerance compression of  $10^{-4}$ . These values represent 23.5%, 51.9%, and 66.7% less than the original data. But, remember that an aggressive compression damages the final RTM result, as we illustrate in Figures 5.16(d), and 5.17(c) for the 2D Marmousi benchmark.

Table 5.7 shows the running time and overhead measurements for the 3D RTM considering the source wavefield compression. Notice that we apply two error tolerances for the lossy compression, which are  $10^{-6}$  and  $10^{-4}$ . Again, it was not possible to get computational measurements for the 3D RTM which implements the source wavefield compression on the VE (SX-Aurora TSUBASA) due its unavailability when the compression tests were done. Thus, the tests in this section consist of running the the 3D

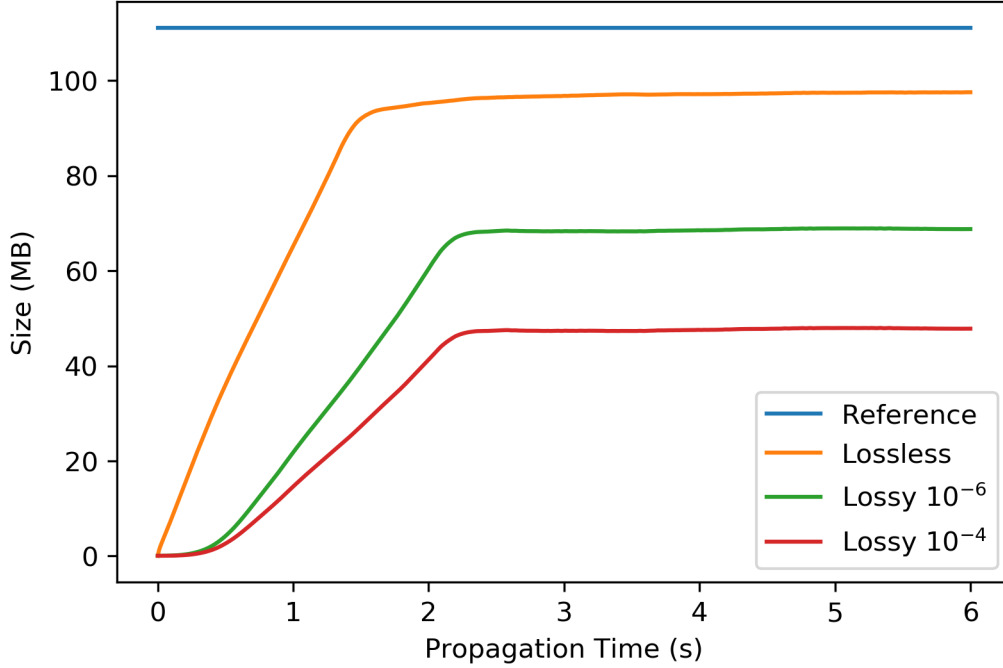


Figure 5.19: Size per time slice for the 3D source wavefield without compression (blue line), with lossless compression (orange line), with a lossy tolerance compression of  $10^{-6}$  (green line), and with a lossy tolerance compression of  $10^{-4}$  (red line).

RTM implementations based on Algorithms 2 and 3 on the CPU (Intel Xeon Ivy Bridge) and GPU (NVIDIA V100) platforms from Santos Dumont cluster. The execution times and overhead values shown in Table 5.7 suggest that the overhead for the source wavefield compression and decompression is high. For instance, the execution time increased 203%, 147%, and 122% when we consider the wavefield compression with the lossless tolerance, lossy error tolerance of  $10^{-6}$ , and lossy error tolerance of  $10^{-4}$  on the CPU platform. Considering the GPU platform, the execution time increased 381%, 245%, and 179% when the wavefield compression is applied with the lossless tolerance, lossy error tolerance of  $10^{-6}$ , and lossy error tolerance of  $10^{-4}$ .

The use of the ZFP library to compress/decompress the source wavefield of the RTM produces sizable overheads for CPU and GPU platforms. However, using a proper platform and compression method, such as the RTM which implements the lossless compression on the GPU platform, we can notice that this version has an execution time 20.2% smaller than the best RTM execution time on the CPU platform. Besides, the RTM with lossless compression demands 23.5% less information to be stored on disk compared to RTM without wavefield compression. Moreover, we can see in Figures 5.16(b) and 5.17(a) the final results have excellent visual quality, where the error compared to the reference outcome is of order  $10^{-6}$ . Assuming the lossy compression of order  $10^{-6}$  the execution time on the GPU platform is 57.2% better than the best RTM execution time

Table 5.7: Time requirements and overhead for the 3D RTM implementation considering different compression rates for the source wavefield. The error tolerance for lossy compressions are set to  $10^{-6}$  and  $10^{-4}$ . The measurements take into account ten executions of each implementation.

Platform	Method	Time (s)[Var. (s)]	Overhead (s)[Var. (s)]
CPU	None	1543.670 [4.108]	-
CPU	Lossless	3130.821 [387.605]	1587.151 [389.555]
CPU	Lossy $10^{-6}$	2261.357 [556.679]	717.687 [561.601]
CPU	Lossy $10^{-4}$	1878.574 [523.458]	334.904 [520.352]
NVIDIA V100	None	256.166 [46.810]	-
NVIDIA V100	Lossless	1232.587 [2.910]	976.421 [3.191]
NVIDIA V100	Lossy $10^{-6}$	882.737 [0.992]	626.571 [0.841]
NVIDIA V100	Lossy $10^{-4}$	713.542 [10.135]	457.376 [8.472]

on the CPU platform and demands 51.9% less information to be stored on disk. Figures 5.16(c) and 5.17(b) support that choice is also a good option for RTM applications.

### 5.3 Uncertainty quantification application

In this section, we describe a workflow for seismic imaging with quantified uncertainty [10, 47] as well as its computational performance. We recall that to access uncertainties is necessary to migrate the recorded data for several hundred (or even thousands) of samples. Migrating the samples makes the computational algorithms not only time but also data-intensive. Even in today’s supercomputers, their implementation and data management are challenging for enabling the implementation of seismic imaging with quantified uncertainty. In this sense, we address BARBOSA et al. [10] that embed the Monte Carlo (MC) sampling method as an outer loop of a larger computational workflow as detailed in Algorithm 9. This algorithm is structured in three sequential stages, enabling a probabilistic framework for seismic imaging.

The first stage of the workflow aims at generating plausible subsurface velocity fields honoring seismic data. Probabilistic inversion, such as Bayesian tomography [10, 12, 19, 20], and stochastic FWI [15, 50, 86, 132, 136] can provide a velocity field ensemble used as input to the second stage. Hence, in Stage 2, an imaging technique migrates the seismogram information using each velocity field sample. This strategy wraps a seismic migration tool into an MC algorithm aiming to build a set of migrated seismic images. We have chosen the RTM as the seismic migration technique to localize the seismic reflectors in the correct depth location in the subsurface [135]. The last stage of the workflow post-processes the ensemble of RTM seismic images, calculating uncertainty maps and extracting features, such as horizons and faults, that characterize uncertainty in the resulting images.

---

**Algorithm 9** Workflow for seismic imaging with quantified uncertainty

---

**Input:** source signals, seismograms, and spatial domain (raw data).

**Output:** ensemble of seismic images.

**Stage 1:** Generate an ensemble of velocity fields:

- Bayesian inversion with simplified physics-based models

**Stage 2:** Propagate uncertainties – migrate the seismograms for the velocity field ensemble using RTM, producing a corresponding ensemble of seismic images

- Monte Carlo loop over samples produced in Stage 1

**Stage 3:** Post-process the RTM seismic images

- Uncertainty maps;
  - Automatic features (horizons) detection;
  - Probabilistic characterization of such features;
- 

Due to its flexibility by design, it is possible to generate different workflow versions, by, for instance, replacing components within the stages (e.g., different strategies for the uncertain velocity fields estimation in Stage 1) targeting to accommodate different demands or efficiency requirements. Nonetheless, we would still be facing a time-consuming computational task in the many-query UQ analysis of Stage 2. That is what motivates us to explore efficient hybrid parallel computational technologies, such as the OpenMP+MPI or OpenACC+MPI, to decrease the reverse time migration execution time. High levels of data compression can also be applied to reduce data transfer among workflow activities and data storage. However, during execution, the RTM application needs to store the source wavefield on disk to build the final seismic image. Storing the source wavefield for the RTM in the uncertainty quantification context is even more challenging because the storage capacity can reach tens of Terabytes of information. An alternative to mitigate the storage demand is to implement the wavefield reconstruction technique as detailed in sections 3.3, 5.1.2, and 5.2.2.

In this context, we detail in the next section the RTM computational implementation within Stage 2 of the MC uncertainty propagation algorithm for two source wavefield management strategies. We will refer to the wrapped RTM into the MC method as Monte Carlo Reverse Time Migration (MC-RTM). Section 5.3.2 presents the numerical experiments, where we expose the execution time requirements, speedups, and hard disk demand for each computational implementation. The last section presents an uncertainty quantification assessing the analysis to highlight the influence of the velocity variability in the seismic images.

### 5.3.1 Reverse time migration under uncertainty

As we wrap RTM into a sampling method in Algorithm 9, for taking into consideration the input uncertainties, the overall computational cost of Stage 2 rises proportionally to the number of samples for the MC method to achieve the statistical convergence,  $N_{MC}$ , the cardinality of the ensemble of possible velocity fields. Typically, seismic raw data recording sets are split into multiple steps ( $N_{shots}$ ) to cope with the potentially high spatial dimensions to be covered and to enhance the signal-to-noise ratio (SNR) in the processing stages. Each step covers fully or partially the imaging domain corresponding to different arrangements of sources and receivers. It is essential to mention that RTM sweeps over the number of shots producing partial migrated images per shot, computed by equation (2.8). When this loop ends, a process called stacking sums the partially migrated seismic images into a single stacked seismic image [62, 125].

Algorithm 10 details the generation of the ensemble of RTM images, where a set of seismograms,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , a set of velocity fields,  $\{\mathbf{v}_1, \dots, \mathbf{v}_{N_{MC}}\}$ , and a seismic source ( $\mathbf{f}$ ) are given as inputs. The indexes  $N_{shots}$  and  $N_{MC}$  represent the number of shots and the number of samples for the MC method. For each MC iteration, we solve the wave equation twice, firstly for the seismic source and secondly for the seismograms associated with it. The computation of the imaging condition uses both solutions (source wavefield, and receiver wavefield), retrieving from persistent storage the source wavefield to build the migrated seismic section and stacking the partial results over time ( $\mathbf{I}_{\Sigma_t}$ ), and over the number of seismograms ( $\mathbf{I}_{\Sigma_{shot\_id}}$ ). At the end of Algorithm 10, we obtain the discrete seismic image set  $\{\mathbf{I}_1, \mathbf{I}_2, \dots, \mathbf{I}_{N_{MC}}\}$  where each  $\mathbf{I}_s$  is a vector in  $\mathbb{R}^3$ . Often, each image is filtered to sharpen the resulting features. We implement the Laplacian filter for that, which consists of partial derivatives of  $\mathbf{I}_s$  as represented by equation (5.6):

$$\nabla^2 \mathbf{I}_s = \frac{\partial^2 \mathbf{I}_s}{\partial x^2} + \frac{\partial^2 \mathbf{I}_s}{\partial y^2} + \frac{\partial^2 \mathbf{I}_s}{\partial z^2} \quad \text{for each } \mathbf{I}_s, \quad (5.6)$$

where  $\nabla^2 = \nabla \cdot \nabla$  is the Laplacian operator.

We show in sections 5.1.2 and 5.2.2 that RTM suffers from persistent storage which can negatively impact its computational performance in terms of storage and execution time. Wavefield compression and reconstruction present themselves as alternative strategies aiming to decrease the storage demand and/or the execution time of RTM algorithms. The results show that the RTM with wavefield reconstruction presents the best values for the storage demand and execution time. Thus, we redesign the RTM under uncertainty in Algorithm 10 to accommodate the RTM with wavefield reconstruction into the MC method. Algorithm 11 details the RTM under uncertainty which takes into account the creation of a random boundary [32] in line 4, and the reconstruction of the source wavefield in line 21 to build the imaging condition in line 22. The proposed modifications

---

**Algorithm 10** Monte Carlo Reverse Time Migration for uncertainty analysis.

---

**Require:**  $\{\mathbf{v}_1, \dots, \mathbf{v}_{N_{MC}}\}$ ,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , and  $\mathbf{f}$ 

```
1: function RTM_UQ( vectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_{N_{MC}}\}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
2:   for  $s = 1$  to  $N_{MC}$  do
3:     read  $\mathbf{v}_s$ ,  $\mathbf{f}$ , and  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ 
4:     initialize image condition  $\mathbf{I}_{\Sigma shot\_id} = 0$ 
5:     for  $shot\_id = 1$  to  $N_{shots}$  do
6:       initialize  $n_t = 0$ 
7:       apply initial conditions for  $i_t = 0$ 
8:       for  $i_t = 1$  to  $N_t$  do
9:          $n_t = i_t * \Delta t$ 
10:        solve equation (2.2) ▷ source wavefield
11:        store  $\mathbf{p}_{n_t}$  for all  $n_t$ 
12:      end for
13:      read  $\mathbf{s}_{shot\_id}$ 
14:      initialize  $n_\tau = 0$ , and  $\mathbf{I}_{\Sigma \tau} = 0$ 
15:      apply initial conditions for  $i_\tau = 0$ 
16:      for  $i_\tau = 1$  to  $N_t$  do
17:         $n_\tau = (N_t - i_\tau) * \Delta \tau$  ▷ reverse time
18:        read  $\mathbf{p}_{n_\tau}$ 
19:        solve equation (2.5) ▷ receiver wavefield
20:        calculate  $\mathbf{I}_{\Sigma n_\tau}^k = \mathbf{I}_{\Sigma n_\tau}^k + (\mathbf{p}_{n_\tau}^k \cdot \bar{\mathbf{p}}_{n_\tau}^k) / (\mathbf{p}_{n_\tau}^k \cdot \mathbf{p}_{n_\tau}^k)$  ▷ imaging condition
21:      end for
22:      stack  $\mathbf{I}_{\Sigma shot\_id}^k = \mathbf{I}_{\Sigma shot\_id}^k + \mathbf{I}_{\Sigma n_\tau}^k$  ▷ stacking
23:    end for
24:     $\mathbf{I}_C^s \leftarrow \mathbf{I}_{\Sigma shot\_id}^k$ 
25:    store  $\mathbf{I}_C^s \forall s \in 1 \leq s \leq N_{MC}$ 
26:  end for
27: end function
```

---

improve the RTM computational performance as described in section 5.1.2, and we intend to achieve the same conclusions for the RTM under uncertainty. The full description of the RTM with wavefield reconstruction can be seen in section 3.3.

---

**Algorithm 11** Improved Monte Carlo Reverse Time Migration for uncertainty analysis.

---

**Require:**  $\{\mathbf{v}_1, \dots, \mathbf{v}_{N_{MC}}\}$ ,  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , and  $\mathbf{f}$

- 1: **function** RTM\_UQ( vectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_{N_{MC}}\}$ , vectors  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$ , vector  $\mathbf{f}$  )
- 2:     **for**  $s = 1$  to  $N_{MC}$  **do**
- 3:         read  $\mathbf{v}_s$ ,  $\mathbf{f}$ , and  $\{\mathbf{s}_1, \dots, \mathbf{s}_{N_{shots}}\}$
- 4:         create a RBC as Algorithm 2 from CLAPP [32]
- 5:         initialize image condition  $\mathbf{I}_{\Sigma shot\_id} = 0$
- 6:         **for**  $shot\_id = 1$  to  $N_{shots}$  **do**
- 7:             initialize  $n_t = 0$
- 8:             apply initial conditions for  $i_t = 0$
- 9:             **for**  $i_t = 1$  to  $N_t$  **do**
- 10:                  $n_t = i_t * \Delta t$
- 11:                 solve equation (2.2) ▷ source wavefield
- 12:                 store  $\mathbf{p}_{n_t}$  for  $N_{t-1}$ , and  $N_t$
- 13:             **end for**
- 14:             initialize  $n_\tau = 0$ , and  $\mathbf{I}_{\Sigma \tau} = 0$
- 15:             read  $\mathbf{s}_{shot\_id}$
- 16:             read  $\mathbf{p}_{N_t}$ , and  $\mathbf{p}_{N_{t-1}}$
- 17:             apply initial conditions for  $i_\tau = 0$
- 18:             **for**  $i_\tau = 1$  to  $N_t$  **do**
- 19:                  $n_\tau = (N_t - i_\tau) * \Delta \tau$  ▷ reverse time
- 20:                 solve equation (2.5) ▷ receiver wavefield
- 21:                 solve equation (3.19) ▷ wavefield reconstruction
- 22:                 calculate  $\mathbf{I}_{\Sigma n_\tau}^k = \mathbf{I}_{\Sigma n_\tau}^k + ([\mathbf{p}_{n_\tau}^R]^k \cdot [\bar{\mathbf{p}}_{n_\tau}^R]^k) / ([\mathbf{p}_{n_\tau}^R]^k \cdot [\mathbf{p}_{n_\tau}^R]^k)$  ▷ imaging condition
- 23:             **end for**
- 24:             stack  $\mathbf{I}_{\Sigma shot\_id}^k = \mathbf{I}_{\Sigma shot\_id}^k + \mathbf{I}_{\Sigma n_\tau}^k$  ▷ stacking
- 25:         **end for**
- 26:          $\mathbf{I}_C^s \leftarrow \mathbf{I}_{\Sigma shot\_id}^k$
- 27:         store  $\mathbf{I}_C^s \forall s \in 1 \leq s \leq N_{MC}$
- 28:     **end for**
- 29: **end function**

---

### 5.3.2 Workflow computational performance analysis

The MC-RTM is based on Algorithms 10 and 11 migrates the seismograms for the velocity field ensemble coming from Bayesian Eikonal Tomography (BET) post-burn-in samples (see Appendix F). These migrations produce a corresponding set of seismic images, where each one has a direct relation with one velocity sample. The RTM embedded into the MC method demands huge computational resources due to the high dimensionality of the stochastic space which depends on the chosen number of samples from the BET post-burn-in phase. We tackle this issue by implementing the MC-RTM on advanced computational technologies, such as multi-CPU and multi-GPU clusters. Therefore, we run the computational implementation of Algorithm 10 on the CPU cluster and the computational implementation of Algorithm 11 on the GPU cluster for further comparison.

Again, we analyze the computational performance in terms of execution time, storage demand, and speedup.

### 5.3.2.1 General computational aspects

First of all, we implement both Algorithms 10 and 11 in C language. The MC-RTM kernel, that is, the second-order wave equation, is discretized with a second-order finite difference scheme in time and an eighth-order scheme in space. Algorithm 10 follows the hybrid computational strategies presented in sections 4.1 and 4.4. In this case, the computational implementation is directed toward the SIMD model (SSE or AVX instructions) to activate vectorization and OpenMP directives to explore multiple cores/threads parallelism for CPU machines. On the other hand, Algorithm 11 follows the hybrid computational strategies described in sections 4.3 and 4.4 where the multi-thread parallelism is activated by OpenACC directives to get the advantage of GPU architectures. Both algorithms apply MPI to manage the execution of multiple migrations and shots. Thus, we broke the total number of migrations into subgroups, and each subgroup is assigned to an MPI process. When a subgroup finishes its workload, the next one starts immediately. The batch execution (an MC-RTM for each subgroup) is repeated until exhausting the number of samples. Besides, each MC-RTM verifies its own Courant–Friedrichs–Lewy (CFL) condition (equation 3.12) to guarantee the stability of the discrete solution of the two-way wave equation for each sample. As we wrap RTM into a sampling method in Algorithms 10 and 11, the overall computational cost rises proportionally to the cardinality of the ensemble of possible velocity fields.

### 5.3.2.2 Computational performance analysis

In this section, we analyze the computational performance for the MC-RTM based on Algorithms 10 and 11. Again, we choose the 2D Marmousi benchmark (see Figure 5.15), where the geometry acquisition follows the same configurations presented in section 5.2.2 for the wavefield compression numerical experiments. But, in this scenario, the grid has  $1574 \times 534$  grid points, where the grid space has 6.25 meters. The grid size considers  $N_a = 50$  except at the top of the velocity model, where we set half of the finite difference stencil length to simulate the free surface. Thus, the tests consist of running the MC-RTM application and calculating the average execution time for ten measurements on CPU and GPU machines for further comparison. The numerical tests consider only one seismogram (a shot). Remember that the CPU machine is an Intel Xeon Ivy Bridge with 24 cores per node, and the GPU machine has an Intel Xeon Skylake CPU with 4 NVIDIA V100 GPUs per node. Both platforms are from the Santos Dumont cluster already described in section 5.1.

Table 5.8 shows the average execution time and the hard disk requirements for the

MC-RTM implementations based on Algorithms 10 and 11. Algorithm 10 requires the full storage of the source wavefield. Nevertheless, instead of storing the source wavefield for every time step ( $\Delta t$ ) based on the FDM, we took advantage of the Nyquist theory [112] to store the wavefield at the Nyquist time step to reduce the amount of information. Therefore, Algorithm 10, which implements the wavefield storage requires 2.638 GB of hard disk on the CPU and GPU machines against 0.005 GB for the wavefield reconstruction implementations (Algorithm 11). This represents  $527.6\times$  less information to be stored.

Table 5.8: Comparison of hard disk and time requirements for the MC-RTM implementation with the wavefield storage and reconstruction methods.

Method	Platform	Hard disk (GB)	Av. Time (s)[Var. (s)]
Storage	CPU	2.638	11.220 [0.147]
Reconstruction	CPU	0.005	6.467 [0.170]
Storage	NVIDIA V100	2.638	7.969 [0.149]
Reconstruction	NVIDIA V100	0.005	3.500 [1.415]

According to the measurements shown in Table 5.8, the GPU implementation with wavefield reconstruction presents the best time execution on average, that is 3.5s against 7.969s of the GPU implementation for Algorithm 10, 6.467s of the CPU cluster for Algorithm 11, and 11.220s of the CPU implementation for Algorithm 10. Consequently, the MC-RTM with wavefield reconstruction is  $2.28\times$  faster than the wavefield storage implementation for the NVIDIA V100,  $2.27\times$  faster than wavefield reconstruction implementation, and 3.21 faster than wavefield storage on the CPU cluster. Remember that Algorithm 11 implements one extra wave equation to reconstruct the source wavefield.

Concerning the speedup calculations, we calculate them only for the MC-RTM based on Algorithm 11 that implements the wavefield reconstruction. The NVIDIA V100 GPU implementation presents the best result as we can see in Figure 5.20. Again, our reference time is the optimized serial MC-RTM code that was executed on the Santos Dumont CPU Cluster. We also run the CPU implementation on the Santos Dumont CPU cluster using 24 CPU cores. Thus, the speedup of OpenMP implementation is 44.0 against 81.3 for the GPU implementation. Therefore, the GPU speedup is  $1.85\times$  better than the CPU speedup.

Due to our need to run the MC-RTM  $n$  times to build the statistics outcomes, the hybrid MC-RTM implementations have to run  $n$  times in  $n$  nodes. In this case, the CPU and GPU times for each node have to be approximately the same. Hence, we run the MC-RTM code for an increasing number of nodes and measure the CPU and GPU times for each one, investigating the weak scalability. We note that the MC-RTM takes, on average, 6.463s with a standard deviation of 0.098s to execute one MC-RTM per CPU node on the Ivy Bridge cluster. Thus, the CPU time for running one RTM in one CPU node is approximately the same as running five RTMs in five CPU nodes (note that  $n = 5$ ). We

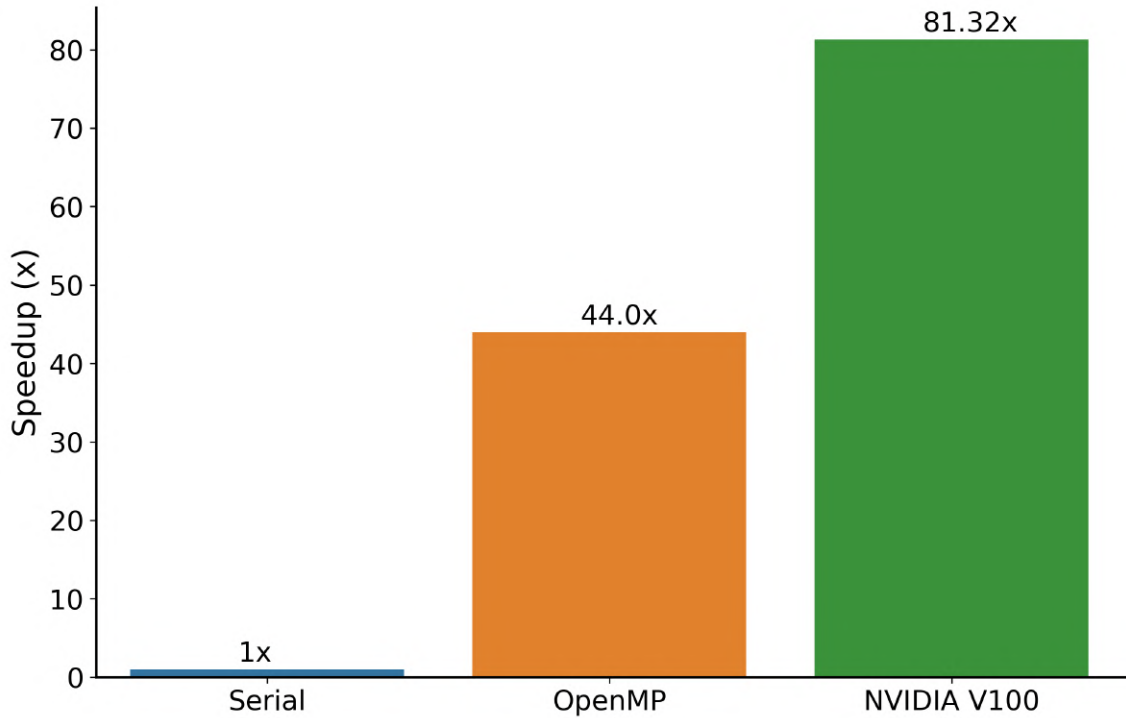


Figure 5.20: MC-RTM speedup across the platforms Santos Dumont CPU Cluster and NVIDIA V100.

observe the same pattern for the GPU platform, that is 3.436s with a standard deviation of 0.786s to execute one MC-RTM per GPU or five MC-RTMs on five GPUs, approximately.

### 5.3.2.3 Hybrid implementation performance comparisons

We run the MC-RTM implementations based on Algorithms 10 and 11 for a set of velocity fields statistically sampled for the BET method. The number of velocity fields used in the migration process comes from the post-burn-in phase of BET. Hence, the number of velocity field samples is  $N_{MC} = 5000$ . The seismograms to be migrated as well as the velocity fields and their spatial dimensions follow the geometry acquisition described in section 5.2.1 for the 2D Marmousi2 benchmark [121].

We run the MC-RTM based on Algorithm 10 on the CPU cluster (Intel Xeon Ivy Bridge) and the MC-RTM based on Algorithm 11 on the CPU-GPU cluster (Intel Xeon Skylake + 4 NVIDIA V100). We set 40 CPU nodes on the CPU cluster, where each CPU node executes two RTMs using 12 cores. On the CPU-GPU node, we set 5 nodes where each one executes four RTMs. Each RTM of the four ones runs in one NVIDIA V100 GPU. Remember that each CPU-GPU node has 4 GPUs.

Table 5.9 show the total execution times of the MC-RTM implementations on the CPU and GPU platforms considering two different wavefield managements. We choose the worst and best scenarios based on Table 5.8. The worst scenario has the highest execution time and storage demand. Opposite to that, the best scenario presents lower

values for the execution time and storage demand. Note that, MC-RTM implementation runs 80 migrations in parallel on the CPU cluster and 20 parallel migrations on the GPU cluster. Nevertheless, the MC-RTM with wavefield reconstruction running on the GPU cluster is  $2.91\times$  faster than the MC-RTM that implements the wavefield storage on the CPU cluster. Furthermore, the MC-RTM with wavefield storage demands 723.86GB and the MC-RTM with wavefield reconstruction needs only 0.26GB, that is,  $2784.01\times$  less storage.

Table 5.9: Comparison of the total execution time for the 2D MC-RTM implementation based on wavefield storage and wavefield reconstruction on CPU and NVIDIA V100 clusters, respectively.

Method	Platform	Total Execution Time
Wavefield Storage	CPU	150h 24min 41.343s
Wavefield Reconstruction	NVIDIA V100	51h 42min 28.567s

### 5.3.3 Uncertainty quantification analysis

We discuss in this section the results of sequentially applying a Bayesian inversion and the MC-RTM to the raw data provided by seismograms generating images with quantified uncertainty as Algorithm 9. In this numerical experiment, we follow BARBOSA *et al.* [10] by using the BET to generate an ensemble of velocity fields<sup>5</sup>. Thus, the uncertainty in the ensemble of velocity fields resulting from BET is preliminary assessed through primary statistics, such as pointwise velocity mean and standard deviation. To isolate the impact of each uncertainty source, we restrict the analysis in the particular Marmousi2 benchmark example to the sensors’ noise (see section 5.2.1 for the acquisition geometry design). Hence, the standard deviation,  $\sigma_d$ , used in the likelihood (equation F.4), is equal to 0.05 seconds. Other components that would induce uncertainty like scarcity of data promoted by acquiring signals only on the surface are not taken into consideration in the velocity estimation stage. That is circumvented by collecting the synthetic data in all grid points. Therefore, noise and model errors are the primary sources of uncertainty in the velocity estimation, and, consequently, generators of uncertainties in the final images. The efficiency of the strategy to extract from the data the dimensionality of the velocity parametrization is expressed in the numbers for this example. The initial number of cells for BET Voronoi’s tessellation is 500. At the end of the process, this number grows to 1000, which is still much less than the total number of grid points. The number of post-burn in velocity fields to be migrated is  $N_{MC} = 5000$ . The number of seismograms is  $N_{shots} = 160$ .

<sup>5</sup>See Appendix F for BET theoretical description and bayes-tomo opensource code in <https://github.com/Uncertainty-Quantification-Project>

Figures 5.21 and 5.22 show the mean and confidence index (equation F.5) for the velocity fields, inputs of the analysis, and for the seismic images, the final outputs. The confidence index expresses the confidence degree associated with the respective fields [71]. We can see that such a criterion assigns, for the velocity estimation, higher confidence ( $> 0.8$ ) to shallow regions. All this information, not present in standard seismic imaging workflows, is deployed to support seismic interpretations. Localized higher variance of the velocity field, particularly in regions of abrupt spatial changes like it occurs in the interface of different geologic formations, might be considered when extracting information from the image, especially if there is also large variability on the seismic amplitudes.

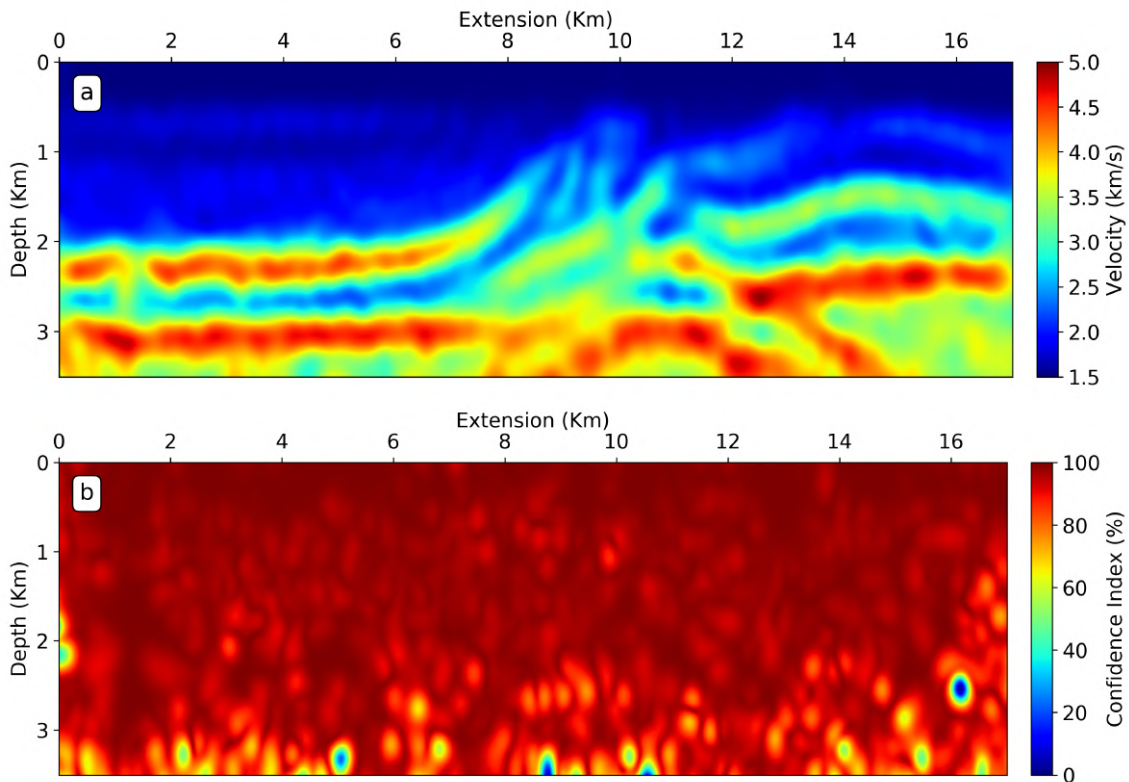


Figure 5.21: Figures (a) and (b) show the mean and confidence index of the Marmousi2 velocity fields.

Figure 5.23 presents the MC-RTM results after the point-wise mean calculation of the outcomes of the seismic image, and the point-wise relative error between Figure 5.23(a) and 5.23(b). Note that the result from Figure 5.23(a) was built based on Algorithm 10, and Algorithm 11 provided the seismic images to calculate the mean seismic image shown in Figure 5.23(b). The results show that the proposed modifications based on the wave-field reconstruction implemented on a multi-GPU cluster did not harm the outcomes. For example, the relative error shown in Figure 5.23(c) assumes values up to 1.79% that are located mainly on the seismic reflector interfaces. Such interfaces represent the transition from positive to negative numerical values and vice-versa.

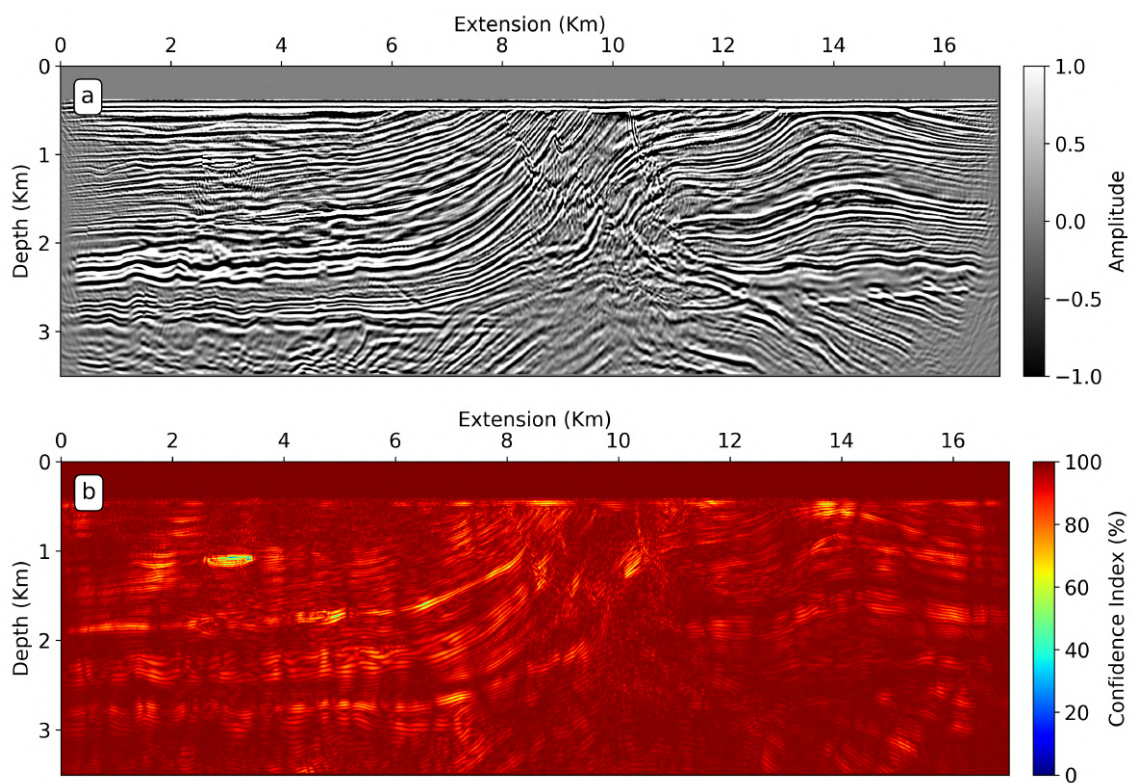


Figure 5.22: Figures (a) and (b) show the mean and confidence index of the migrated seismic images.

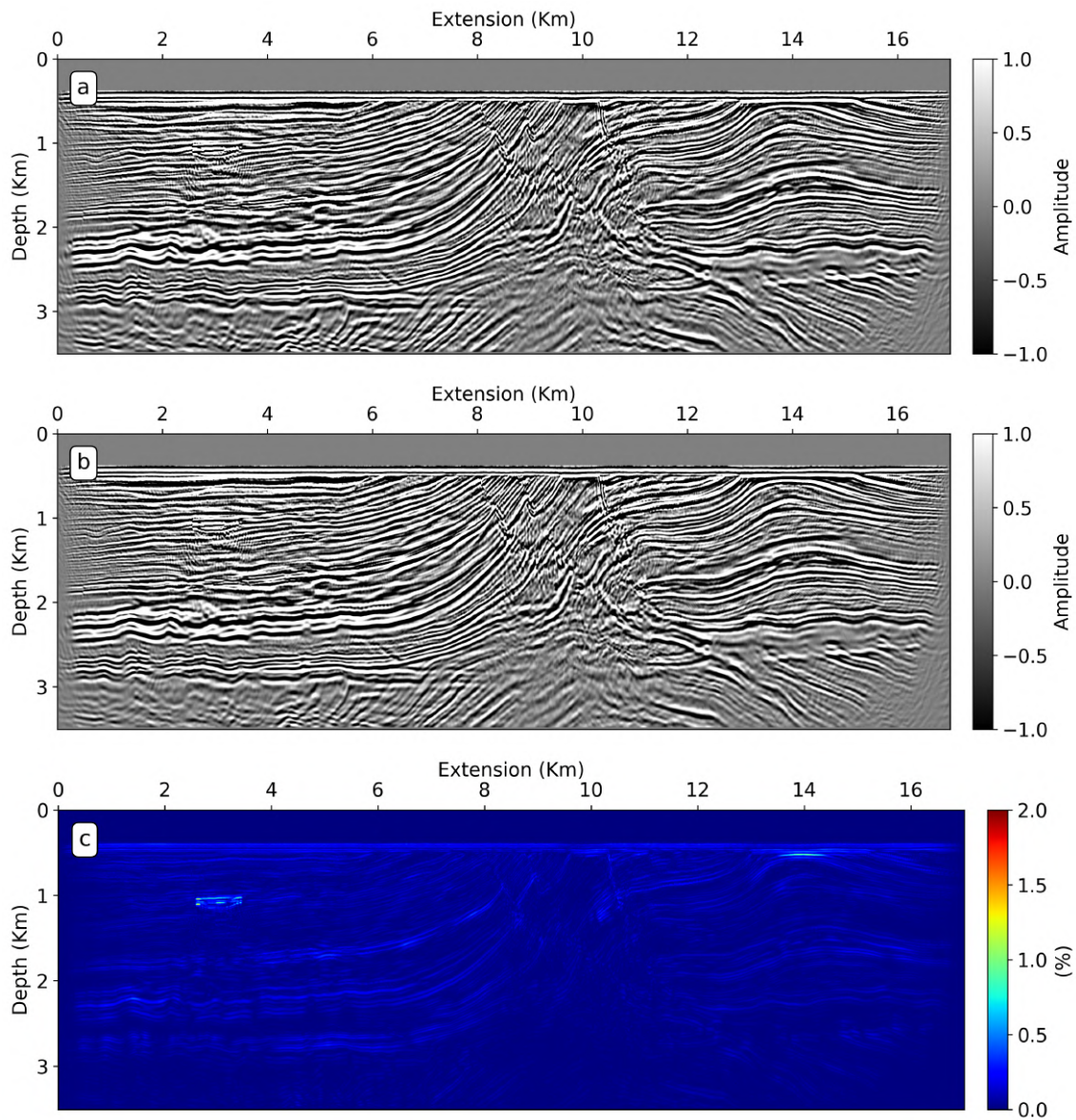


Figure 5.23: Comparison between mean migrated seismic image built by the RTM with storage (a) and the RTM with wavefield reconstruction (b). Figure (c) shows relative error between seismic images (a) and (b).

# Chapter 6

## Final Remarks and Perspectives

The present work addresses algorithmic advances for improving RTM's computational efficiency, which can considerably impact the processing time of geophysical applications and their disk requirements. The study focus on the impact of computational resource usage based on different RTM algorithm strategies, which leads mainly to seismogram and wavefield storage. Strategies to manage the storage requires, in general, numerical calculations beyond the normally used by RTM conventional implementations. Hence, it is vital in today's supercomputers to provide efficient implementations aiming to explore hybrid parallelism levels that range from vectorization (lowest level) to nodes connected by a network (highest level), keeping code portability as much as possible.

The storage management and efficient implementations for RTM are tackled by introducing different techniques to reduce storage and exploring a range of heterogeneous supercomputers. Specifically, this work explores algorithm strategies based on decimation, compression, and reconstruction methods to reduce storage required on methods based on wave equations such as RTM. Besides, the RTM algorithms are implemented in C language, using vectorization, OpenMP, OpenACC, and MPI to explore the best features of supercomputers equipped with multi-core CPUs, GPUs, and VPs.

The RTM storage management considers decimation, compression, and reconstruction methods. One strategy for the RTM algorithm combines decimation and compression of the source wavefield. The decimation process must respect the Nyquist theory to avoid temporal aliasing, and the compression uses the ZFP library. A second strategy introduces an extra wave equation to the RTM problem to reconstruct the source wavefield based on initial and boundary conditions. The initial condition uses the final states of the RTM forward-propagated problem after solving it. The introduction of the extra wave equation is practical only if all energy is kept inside the domain. Thus, ABCs are not used because they attenuate the wave signals around the domain. Instead, RBCs are implemented to generate a random wavefield on the boundaries avoiding its attenuation. This process preserves the wavefield energy allowing its complete reconstruction. Further, the random wavefield has a minimum impact on the seismic image construction, as suggested by the

numerical experiments.

The new RTM algorithm strategies are compared to the conventional ones. In this work, we assume the conventional strategy, based on the total storage of the source wavefield, to further build the seismic image by recovering it from the disk. In 3D scenarios, it is possible to reduce  $3.01\times$  the wavefield storage assuming aggressive lossy compression rates such as the fixed-accuracy error tolerance  $10^{-6}$ . Without tuning, the lossless compression reduces the wavefield storage  $1.31\times$ . The tolerance defined by the lossless and lossy compression impacts the image quality. The seismic image results present errors of order  $10^{-3}$  for the most aggressive compression case and errors of order  $10^{-6}$  for the lossless compression. Savings in disk requirements are even better when using wavefield reconstruction. The 3D RTM with wavefield reconstruction demands  $300.83\times$  less storage than the 3D RTM that implements the wavefield storage. Once we analyze the nominal values for multiple shots being executed in parallel, the presented values are enormous. Considering 16 shots running in parallel for the 3D seismic survey presented in the numerical experiments, the RTM with wavefield storage demands 2112.992 GB, RTM with wavefield lossless compression demands 797.12 GB, RTM with wavefield lossy compression  $10^{-6}$  requires 501.44 GB, RTM with wavefield lossy compression  $10^{-4}$  demands 346.88 GB, and RTM with wavefield reconstruction demands 7.024 GB. Hence, the RTM algorithm strategy plays an essential role in saving disk storage and data transfer among the host, device (such as GPUs and VPs), and disk. Many-query computations, like those found in uncertainty quantification, also benefit from the same storage strategies. For instance, the embedded RTM into the MC method reduces  $2784.01\times$  the storage demand considering the wavefield reconstruction method.

Efficient and portable implementations for the RTM are also explored to evaluate the computational performance when strategies to reduce storage are taken into account. Again, the reference RTM implements the wavefield storage running on a multi-CPU cluster equipped with 24 physical cores. Incorporating the ZFP library to compress the source wavefield raises the RTM execution time in 203%, 147%, and 122% for the lossless tolerance, lossy error tolerance  $10^{-6}$ , and lossy error tolerance  $10^{-4}$  on the CPU cluster, and 381%, 245%, and 179% on the GPU cluster, respectively. However, the storage demand decreased 23.5%, 51.9%, and 66.7% on both CPU and GPU platforms, considering the same error tolerances. Choosing the compression strategy to reduce the wavefield storage requires better platforms to compensate for the overhead, such as multi-GPU clusters. For instance, assuming the lossy compression of order  $10^{-6}$  the execution time on the GPU platform is  $1.75\times$  better than the best RTM execution time on the CPU platform and demands 51.9% less information to be stored on disk. Instead of storing the source wavefield, its reconstruction positively impacts the RTM execution time and disk requirement. RTM with wavefield reconstruction on the GPU platform performs  $1.83\times$  better than the RTM with wavefield storage on the same platform demanding  $300.83\times$  less information

to be stored. Although not used in all numerical tests, the RTM implementation on the VP platform also benefits from the wavefield reconstruction method reaching the best execution time of 108.582s with 8 vector cores for the 3-D numerical experiments. Last but not least, the MC-RTM with wavefield reconstruction implemented on the GPU platform allowed reducing its total execution time  $2.91\times$  compared to MC-RTM with wavefield storage on the CPU platform. More important, with a three orders of magnitude reduction in persistent storage. In this scenario, we allocate 5 nodes on the GPU cluster and 40 nodes on the CPU cluster. Nevertheless, reaching accuracy and fast solutions demanding less computational resources was possible.

Energy consumption has become a prime concern over the years due to the environmental impact of generating it. Because of that, the scientific community has pursued better ways to provide clean energy and developed efficient energy and hardware usage algorithms. The GREEN500<sup>1</sup> is an example of measuring the computational performance per watt of TOP500<sup>2</sup> supercomputer list for high-performance LINPACK benchmarks at double-precision floating-point format. Although metrics to measure the computational and energy efficiency are not discussed in this work, the RTM with wavefield reconstruction strategy implemented on co-processors like multi-GPUs and multi-VPs indicates high hardware usability. The profiling reports shown in Appendix G supports that by detailing the usability of the co-processors, vectorization rate, and cache missing rate. The computational performance measurements show that it was possible to achieve accurate results by spending almost  $3\times$  less in execution time and reducing the disk requirement of  $O(10^3)$ . Furthermore, the RTM with wavefield reconstruction allows allocating less computational resources due the disk requirements reduction.

In this context, future research can leverage the RTM algorithms to obtain fast and accurate wave equation solutions (source and receiver wavefields), building upon optimized stencil calculations, wavefield, seismogram, and partial seismic images storage management, and efficient implementations for the emerging hardware technologies. These strategies impact the RTM algorithm based on the cross-correlation imaging conditions (ICs) and RTMs, which explore alternative ICs such as extended ICs, wavefield decomposition ICs, angle domain ICs, and others. Each has its own implementation complexity and can demand even more computational resources to be applied, such as the RTM with the wavefield decomposition IC. Beyond the classical RTM formulations, iterative approaches like the MC-RTM have particularities to be treated to deal with the computational cost that rises proportionally to the number of iterations. For instance, MC-RTM explores a multi-level parallelism strategy based on vectorization, multi-core and/or multi-GPU parallelism and multi-node parallelism. These strategies act on the computational cost per iteration, but the total number of them can be decreased by providing

---

<sup>1</sup><https://www.top500.org/lists/green500/>

<sup>2</sup><https://www.top500.org/>

partial solutions of RTM as training data for machine/deep learning approaches build the full solution [47] or even diminishing the redundancy of shot gathers [107]. It is possible to extend these advances to seismic imaging techniques based on wave equations like LS-RTM, FWI, and FWI Imaging.

Depending on the geological scenario, the basis of RTM, LS-RTM, and FWI need to adequately the governing wave equations. For instance, a significant presence of seismic anisotropy requires anisotropic elastic wave equations to correctly image the subsurface structures. However, such equations impose challenges to deal with P-wave and S-wave propagations in anisotropic media as well as abundant computational resources to simulate the elastic wavefield [135]. An alternative strategy to mitigate these challenges is to use acoustic anisotropic methods [7, 11, 46]. Formulations of acoustic anisotropic wave equations are simpler than elastic ones. Besides, they provide accurate solutions for different anisotropic scenarios, such as vertical and tilted transversely isotropic media (VTI and TTI). Efficient stencil calculations aligned to methods to mitigate the inherent memory-bound problem of seismic imaging methods based on wave equations are essential to optimize the computational effort to compute the anisotropic solutions [83].

One objective of this work is to introduce algorithms for RTM and measure their computational performance on heterogeneous supercomputers. Thus, the present RTM algorithms take advantage of vectorization, OpenMP, OpenACC, and MPI to explore different levels of parallelism. The RTM with the wavefield reconstruction is an example of that. However, we know that RTM's computational implementations can be tuned by improving the cache memory usage and load balancing, exploiting the thread and data mapping, reducing data transfer, and exploiting loop transformations [100, 104, 105]. Tuning the ZFP compressor and exploring other compression techniques deserve better attention. For example, the ZFP compression added around 3.377s on average to compress each seismogram and around 600.789s to compress the source wavefield per shot, both for the 3D numerical experiments. These values are measurements taken on the GPU cluster and seem to be huge, but considering the compression per time step, where  $N_t = 600$ , the values are 0.0056s to compress a vector of  $320 \times 320$  grid points for the seismogram and 1.445s to compress a vector of  $501 \times 501 \times 235$  grid points for the source wavefield. The time evolution nature of the RTM formulation adds an extra complexity layer to compression methods. Another alternative is to explore other compression techniques such as Gzip, ZLIB, SZ, and MGARD libraries [5, 37, 38, 49, 74, 131]. Except for the MGARD, all of them are general purpose [5]. In contrast, MGARD is a technique based on the theory of multi-grid methods designed to lossy compress scientific data. Besides, the MGARD performs the compression on different grid representation levels allowing the use of nonuniform grids in each direction [5, 6, 74], and provides implementations that explore parallel architectures like GPUs<sup>3</sup>.

---

<sup>3</sup><https://github.com/CODARcode/MGARD>

# References

- [1] ABAUNZA, V., 2018, Performance Optimization of Geophysics Stencils on HPC Architectures. Tese de Doutorado, Federal University of Rio Grande do Sul, Brazil.
- [2] ADRIANO, M. S., FIGUEIREDO, J. P., COELHO, P. H. G. R., et al., 2022, “Tectonic and stratigraphic evolution of the Santos Basin rift phase: New insights from seismic interpretation on Tupi oil field area”, Journal of South American Earth Sciences, v. 116, pp. 103842.
- [3] AFANASYEV, I. V., VOEVODIN, V. V., VOEVODIN, V. V., et al., 2019, “Analysis of relationship between SIMD-processing features used in NVIDIA GPUs and NEC SX-Aurora TSUBASA vector processors”. In: International Conference on Parallel Computing Technologies, pp. 125–139. Springer, July.
- [4] AINSWORTH, M., KLASKY, S., WHITNEY, B., 2017, “Compression using lossless decimation: analysis and application”, SIAM Journal on Scientific Computing, v. 39, n. 4, pp. B732–B757.
- [5] AINSWORTH, M., TUGLUK, O., WHITNEY, B., et al., 2019, “Multilevel techniques for compression and reduction of scientific data—The multivariate case”, SIAM Journal on Scientific Computing, v. 41, n. 2, pp. A1278–A1303.
- [6] AINSWORTH, M., TUGLUK, O., WHITNEY, B., et al., 2019, “Multilevel techniques for compression and reduction of scientific data-quantitative control of accuracy in derived quantities”, SIAM Journal on Scientific Computing, v. 41, n. 4, pp. A2146–A2171.
- [7] ALKHALIFAH, T., 2000, “An acoustic wave equation for anisotropic media”, Geophysics, v. 65, n. 4, pp. 1239–1250.
- [8] BARBOSA, C. H. S., COUTINHO, A. L. G. A., 2022, “Multi-GPU 3-D Reverse Time Migration with Minimum I/O”. In: Navaux, P., Barrios H., C. J., Os-thoff, C., et al. (Eds.), High Performance Computing, pp. 160–173, Cham, September. Springer International Publishing. ISBN: 978-3-031-23821-5.

- [9] BARBOSA, C. H., COUTINHO, A. L., 2020, “Enhancing Reverse Time Migration: Hybrid Parallelism plus Data Compression”. In: Proceedings of the XLI Ibero-Latin-American Congress on Computational Methods in Engineering. ABMEC, November.
- [10] BARBOSA, C. H., KUNSTMANN, L. N., SILVA, R. M., et al., 2020, “A workflow for seismic imaging with quantified uncertainty”, Computers & Geosciences, v. 145, pp. 104615.
- [11] BARBOSA, C., DA SILVA, J., DIAS, B., et al., 2017, “An Analysis of the Reverse Time Migration in a Transversely Isotropic Media”. In: 79th EAGE Conference and Exhibition 2017, v. 2017, pp. 1–5. European Association of Geoscientists & Engineers, June.
- [12] BELHADJ, J., ROMARY, T., GESRET, A., et al., 2018, “New parametrizations for bayesian seismic tomography”, Inverse Problems, v. 34, pp. 33.
- [13] BERENGER, J.-P., 1994, “A perfectly matched layer for the absorption of electromagnetic waves”, Journal of computational physics, v. 114, n. 2, pp. 185–200.
- [14] BERKHOUT, A., 1981, “Wave field extrapolation techniques in seismic migration, a tutorial”, Geophysics, v. 46, n. 12, pp. 1638–1656.
- [15] BISWAS, R., SEN, M., 2017, “2D full-waveform inversion and uncertainty estimation using the reversible jump Hamiltonian Monte Carlo”. In: SEG Technical Program Expanded Abstracts 2017, Society of Exploration Geophysicists, pp. 1280–1285.
- [16] BODIN, T., SAMBRIDGE, M., 2009, “Seismic tomography with reversible jump algorithm”, Geophysical Journal International, v. 178, pp. 1411–1436.
- [17] BOEHM, C., HANZICH, M., DE LA PUENTE, J., et al., 2016, “Wavefield compression for adjoint methods in full-waveform inversion”, Geophysics, v. 81, n. 6, pp. R385–R397.
- [18] BORDING, R. P., 2004, “Finite difference modeling-nearly optimal sponge boundary conditions”. In: 2004 SEG Annual Meeting. OnePetro, October.
- [19] BOTERO, A., GESRET, A., ROMARY, T., et al., 2016, “Stochastic seismic tomography by interacting Markov chains”, Geophysical Journal International, v. 207, pp. 374–392.
- [20] BRANTUT, N., 2018, “Time-resolved tomography using acoustic emissions in the laboratory, and application to sandstone compaction”, Geophysical Journal International, v. 213, pp. 2177–2192.

- [21] BUCHTY, R., HEUVELINE, V., KARL, W., et al., 2012, “A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators”, Concurrency and Computation: Practice and Experience, v. 24, n. 7, pp. 663–675.
- [22] BURGESS, T. J., 2017, Computational and numerical aspects of full waveform seismic inversion. Tese de Doutorado, Imperial College, London. Disponível em: <<https://spiral.imperial.ac.uk/handle/10044/1/65658>>.
- [23] CARNEIRO, M. S. R., DIAS, B. P., SOARES FILHO, D. M., et al., 2018, “On the Scaling of the Update Direction for Multi-parameter Full Waveform Inversion: Applications to 2D Acoustic and Elastic Cases”, Pure and Applied Geophysics, v. 175, pp. 217–241. doi: 10.1007/s00024-017-1677-9.
- [24] CEBRIAN, J. M., NATVIG, L., JAHRE, M., 2020, “Scalability analysis of AVX-512 extensions”, The journal of supercomputing, v. 76, n. 3, pp. 2082–2097.
- [25] CERJAN, C., KOSLOFF, D., KOSLOFF, R., et al., 1985, “A nonreflecting boundary condition for discrete acoustic and elastic wave equations”, Geophysics, v. 50, n. 4, pp. 705–708.
- [26] CHANDRASEKARAN, S., JUCKELAND, G., 2017, OpenACC for Programmers: Concepts and Strategies. Addison-Wesley Professional.
- [27] CHEN, G., PENG, Z., LI, Y., 2022, “A framework for automatically choosing the optimal parameters of finite-difference scheme in the acoustic wave modeling”, Computers & Geosciences, v. 159, pp. 104948.
- [28] CHU, C., STOFFA, P. L., 2012, “Determination of finite-difference weights using scaled binomial windows”, Geophysics, v. 77, n. 3, pp. W17–W26.
- [29] CHU, C., STOFFA, P. L., SEIF, R., 2009, “High-order rotated staggered finite difference modeling of 3D elastic wave propagation in general anisotropic media”. In: SEG Technical Program Expanded Abstracts 2009, Society of Exploration Geophysicists, pp. 291–295.
- [30] CLAERBOUT, J. F., 1984, Imaging the earth’s interior, v. 1. Stanford, California, Stanford University Stanford.
- [31] CLAPP, R. G., 2008, “Reverse time migration: Saving the boundaries”, Stanford Exploration Project, v. 137.
- [32] CLAPP, R. G., 2009, “Reverse time migration with random boundaries”. In: Seg technical program expanded abstracts 2009, Society of Exploration Geophysicists, pp. 2809–2813.

- [33] CRASE, E., PICA, A., NOBLE, M., et al., 1990, “Robust elastic nonlinear waveform inversion: Application to real data”, Geophysics, v. 55, n. 5, pp. 527–538.
- [34] DAVY, R. G., FRAHM, L., BELL, R., et al., 2021, “Generating High-Fidelity Reflection Images Directly From Full-Waveform Inversion: Hikurangi Subduction Zone Case Study”, Geophysical Research Letters, v. 48, n. 19, pp. e2021GL094981.
- [35] DE LA PUENTE, J., 2015, D6.3 Website deploying a suite of geophysical tests for wave propagation problems on extreme scale machines. HPC4E - High Performance Computing for Energy 689772, HPC4E Consortium Partners.
- [36] DE MEDEIROS LARA, M. A., 2012, Paralelização de Um Algoritmo de Propagação da Onda Acústica 2D Usando MPI. Tese de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Setembro. Disponível em: <http://www.coc.ufrj.br/pt/dissertacoes-de-mestrado/112-msc-pt-2012/2268-matheus-alves-de-medeiros-lara>.
- [37] DEUTSCH, P., 1996. “GZIP File Format Specification Version 4.3”. <https://www.rfc-editor.org/info/rfc1952>. Accessed: 2022-06-29.
- [38] DI, S., CAPPELLO, F., 2016, “Fast error-bounded lossy HPC data compression with SZ”. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 730–739. IEEE, May.
- [39] DI BARTOLO, L., DORS, C., MANSUR, W. J., 2012, “A new family of finite-difference schemes to solve the heterogeneous acoustic wave equation new finite-difference schemes for acoustics”, Geophysics, v. 77, n. 5, pp. T187–T199.
- [40] DIFFENDERFER, J., FOX, A. L., HITTINGER, J. A., et al., 2019, “Error analysis of zfp compression for floating-point data”, SIAM Journal on Scientific Computing, v. 41, n. 3, pp. A1867–A1898.
- [41] DONGARRA, J., FOSTER, I., FOX, G., et al., 2003, Sourcebook of parallel computing, v. 3003. Morgan Kaufmann Publishers San Francisco.
- [42] EGAWA, R., FUJIMOTO, S., YAMASHITA, T., et al., 2020, “Exploiting the potentials of the second generation SX-Aurora TSUBASA”. In: 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), pp. 39–49. IEEE, November.
- [43] ETIENNE, V., MOMIN, A., GATINEAU, L., et al., 2021, “Performance Characterization of a Vector Architecture for Seismic Applications”. In: Fifth EAGE

Workshop on High Performance Computing for Upstream, pp. 1–5. European Association of Geoscientists & Engineers, September.

- [44] FARIA, E. L., 1986, Migração Antes do Empilhamento Utilizando Propagação Reversa no Tempo. Tese de Mestrado, Universidade Federal da Bahia, Bahia, Janeiro. Disponível em: <<http://www.cpgg.ufba.br/pppginfo/resumos.new/gfm/gfm0056a.html>>.
- [45] FELDSPAR, A., 1997. “An explanation of the DEFLATE Algorithm”. <https://zlib.net/feldspar.html>. Accessed: 2022-06-29.
- [46] FLETCHER, R. P., DU, X., FOWLER, P. J., 2009, “Reverse time migration in tilted transversely isotropic (TTI) media”, Geophysics, v. 74, n. 6, pp. WCA179–WCA187.
- [47] FREITAS, R. S., BARBOSA, C. H., GUERRA, G. M., et al., 2021, “An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty”, Computational Geosciences, v. 25, n. 3, pp. 1229–1250.
- [48] FUKAYA, T., IWASHITA, T., 2018, “Time-space tiling with tile-level parallelism for the 3D FDTD method”. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, pp. 116–126, January.
- [49] GAILLY, J.-L., ADLER, M., 2004, “Zlib compression library”, [Other]. Disponível em: <<http://www.dspace.cam.ac.uk/handle/1810/3486>>.
- [50] GEBRAAD, L., BOEHM, C., FICHTNER, A., 2020, “Bayesian elastic full-waveform inversion using Hamiltonian Monte Carlo”, Journal of Geophysical Research: Solid Earth, v. 125, n. 3, pp. e2019JB018428.
- [51] GIROTTO, I., FARBER, R. M., 2012, “Multi-threaded architectures: Evolution, costs, opportunities”. In: Handbook of Research on Computational Science and Engineering: Theory and Practice, IGI Global, pp. 22–47.
- [52] GIVOLI, D., 2014, “Time reversal as a computational tool in acoustics and elastodynamics”, Journal of Computational Acoustics, v. 22, n. 03, pp. 1430001.
- [53] GREEN, P. J., 1995, “Reversible jump Markov chain Monte Carlo computation and Bayesian model determination”, Biometrika, v. 82, pp. 711–732.
- [54] GREEN, P. J., HASTIE, D. I., 2009, “Reversible jump MCMC”, Genetics, v. 155, pp. 1391–1403.

- [55] GUERMOUCHE, A., ORGERIE, A.-C., 2019, Experimental analysis of vectorized instructions impact on energy and power consumption under thermal design power constraints. Relatório Técnico hal-02167083v2, HAL open science. Disponível em: <<https://hal.archives-ouvertes.fr/hal-02167083/>>.
- [56] HEATH, M. T., 2015, “A tale of two laws”, The International Journal of High Performance Computing Applications, v. 29, n. 3, pp. 320–330.
- [57] HUANG, R., ZHANG, Z., WU, Z., et al., 2021, “Full-waveform inversion for full-wavefield imaging: Decades in the making”, The Leading Edge, v. 40, n. 5, pp. 324–334.
- [58] JACQUELIN, M., ARAYA-POLO, M., MENG, J., 2022, “Scalable distributed high-order stencil computations”. In: 2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 411–423. IEEE Computer Society, November.
- [59] JONES, I. F., 2010, An introduction to velocity model building. 1st ed. , EAGE Publications.
- [60] JONES, I. F., BLOOR, R. I., BIONDI, B. L., et al., 2008, Prestack depth migration and velocity model building. Tulsa, OK 74170-2740, Society of Exploration Geophysicists.
- [61] JORGE, S. C., 2016, Explorando Paralelismo Híbrido na Propagação da Onda Acústica 3D. Tese de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Setembro. Disponível em: <<http://www.coc.ufrj.br/pt/dissertacoes-de-mestrado/389-msc-pt-2016/8312-schirley-correa-jor>>.
- [62] KEAREY, P., BROOKS, M., HILL, I., 2013, An introduction to geophysical exploration. John Wiley & Sons.
- [63] KOMATITSCH, D., MARTIN, R., 2007, “An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation”, Geophysics, v. 72, n. 5, pp. SM155–SM167.
- [64] KOMATSU, K., MOMOSE, S., ISOBE, Y., et al., 2018, “Performance evaluation of a vector supercomputer SX-Aurora TSUBASA”. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 685–696. IEEE, November.

- [65] KUKREJA, N., HÜCKELHEIM, J., LOUBOUTIN, M., et al., 2019, “Combining checkpointing and data compression to accelerate adjoint-based optimization problems”. In: European Conference on Parallel Processing, pp. 87–100. Springer, August.
- [66] KUKREJA, N., HÜCKELHEIM, J., LOUBOUTIN, M., et al., 2022, “Lossy checkpoint compression in full waveform inversion: a case study with ZFPv0. 5.5 and the overthrust model”, Geoscientific Model Development, v. 15, n. 9, pp. 3815–3829.
- [67] KUSHIDA, N., LIN, Y.-T., NIELSEN, P., et al., 2019, “Acceleration in Acoustic Wave Propagation Modelling Using OpenACC/OpenMP and Its Hybrid for the Global Monitoring System”. In: International Workshop on Accelerator Programming Using Directives, pp. 25–46. Springer, June.
- [68] LAILLY, P., 1983, “The seismic inverse problem as a sequence of before stack migrations”. In: Conference on inverser scattering: Theory and Application, pp. 206–220, SIAM.
- [69] LAILLY, P., SANTOSA, F., 1984, “Migration methods: partial but efficient solutions to the seismic inverse problem”, Inverse problems of acoustic and elastic waves, v. 51, pp. 1387–1403.
- [70] LEVEQUE, R. J., 2007, Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems. SIAM.
- [71] LI, L., CAERS, J., SAVA, P., 2015, “Assessing seismic uncertainty via geostatistical velocity-model perturbation and image registration: An application to subsalt imaging”, The Leading Edge, v. 34, pp. 1064–1070.
- [72] LI, Q., FU, L.-Y., WU, R.-S., et al., 2020, “Efficient Acoustic Reverse Time Migration With an Attenuated and Reversible Random Boundary”, IEEE Access, v. 8, pp. 34598–34610.
- [73] LI, Y., SUN, J., 2016, “3D magnetization inversion using fuzzy c-means clustering with application to geology differentiation”, Geophysics, v. 81, pp. J61–J78.
- [74] LIANG, X., WHITNEY, B., CHEN, J., et al., 2021, “Mgard+: Optimizing multi-level methods for error-bounded scientific data reduction”, IEEE Transactions on Computers, v. 71, n. 7, pp. 1522–1536.
- [75] LINDSTROM, P., 2014, “Fixed-rate compressed floating-point arrays”, IEEE transactions on visualization and computer graphics, v. 20, n. 12, pp. 2674–2683.

- [76] LINDSTROM, P., CHEN, P., LEE, E.-J., 2016, “Reducing disk storage of full-3D seismic waveform tomography (F3DT) through lossy online compression”, Computers & Geosciences, v. 93, pp. 45–54.
- [77] LINDSTROM, P., CHEN, P., LEE, E.-J., 2016, “Reducing disk storage of full-3D seismic waveform tomography (F3DT) through lossy online compression”, Computers & Geosciences, v. 93, pp. 45–54.
- [78] LIU, D. C., NOCEDAL, J., 1989, “On the limited memory BFGS method for large scale optimization”, Mathematical programming, v. 45, n. 1, pp. 503–528.
- [79] LIU, H., LI, B., LIU, H., et al., 2012, “The issues of prestack reverse time migration and solutions with graphic processing unit implementation”, Geophysical Prospecting, v. 60, n. 5, pp. 906–918.
- [80] LIU, X., LIU, Y., 2018, “Plane-wave domain least-squares reverse time migration with free-surface multiples”, Geophysics, v. 83, n. 6, pp. S477–S487.
- [81] LOGG, A., MARDAL, K.-A., WELLS, G., 2012, Automated solution of differential equations by the finite element method: The FEniCS book, v. 84. Springer Science & Business Media.
- [82] LOUBOUTIN, M., LANGE, M., LUPORINI, F., et al., 2019, “Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration”, Geosci. Model Dev., v. 12, pp. 1165–1187.
- [83] LOUBOUTIN, M., LANGE, M., HERRMANN, F. J., et al., 2017, “Performance prediction of finite-difference solvers for different computer architectures”, Computers & Geosciences, v. 105, pp. 148–157.
- [84] LOUBOUTIN, M., LANGE, M., LUPORINI, F., et al., 2019, “Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration”, Geoscientific Model Development, v. 12, n. 3, pp. 1165–1187.
- [85] MACIÀ, S., MATEO, S., MARTÍNEZ-FERRER, P. J., et al., 2018, “Saiph: Towards a dsl for high-performance computational fluid dynamics”. In: Proceedings of the Real World Domain Specific Languages Workshop 2018, pp. 1–10, February.
- [86] MARTIN, J., WILCOX, L. C., BURSTEDDE, C., et al., 2012, “A stochastic Newton MCMC method for large-scale statistical inverse problems with application to seismic inversion”, SIAM Journal on Scientific Computing, v. 34, n. 3, pp. A1460–A1487.

- [87] MATHUR, R., ATCHESON, R., KUBO, Y., 2020, “Optimizations for seismic applications on the NEC SX-Aurora TSUBASA”. In: SEG International Exposition and Annual Meeting. OnePetro, October.
- [88] MÉTIVIER, L., BROSSIER, R., VIRIEUX, J., et al., 2012, “The truncated Newton method for full waveform inversion”. In: 2012 SEG Annual Meeting. OnePetro, November.
- [89] MICHELS, F., SERPA, M., CARASTAN-SANTOS, D., et al., 2021, “Técnicas de otimização em Aceleradores Vetoriais NEC SX-Aurora”. In: Anais da XXI Escola Regional de Alto Desempenho da Região Sul, pp. 107–108. SBC.
- [90] MOMOSE, S., KUBO, Y., IKUTA, M., 2021, “Performance Evaluation of Stencil Calculation in RTM Code”. In: Fifth EAGE Workshop on High Performance Computing for Upstream, pp. 1–5. European Association of Geoscientists & Engineers, September.
- [91] MUDALIGE, G. R., REGULY, I., JAMMY, S. P., et al., 2019, “Large-scale performance of a DSL-based multi-block structured-mesh application for Direct Numerical Simulation”, Journal of Parallel and Distributed Computing, v. 131, pp. 130–146.
- [92] MUFTI, I. R., 1990, “Large-scale three-dimensional seismic models and their interpretive significance”, Geophysics, v. 55, n. 9, pp. 1166–1182.
- [93] NGUYEN, B. D., MCMECHAN, G. A., 2015, “Five ways to avoid storing source wavefield snapshots in 2D elastic prestack reverse time migration”, Geophysics, v. 80, n. 1, pp. S1–S18.
- [94] NOCEDAL, J., WRIGHT, S. J., 2006, Numerical Optimization. New York, Springer.
- [95] NOGUEIRA, P., PORSANI, M. J., 2021, “3D reverse time migration using a wavefield domain dynamic approach”, Journal of Applied Geophysics, v. 190, pp. 104345.
- [96] NOGUEIRA, P., LEITE, V., PORSANI, M. J., 2020, “A wavefield domain dynamic approach: Application in reverse time migration”, Journal of Applied Geophysics, v. 177, pp. 104036.
- [97] PASALIC, D., MCGARRY, R., 2010, “Convolutional perfectly matched layer for isotropic and anisotropic acoustic wave equations”. In: SEG Technical Program Expanded Abstracts 2010, Society of Exploration Geophysicists, pp. 2925–2929.

- [98] PERSHIN, I. S., LEVCHENKO, V. D., PEREPELKINA, A. Y., 2019, “Performance limits study of stencil codes on modern GPGPUs”, Supercomputing Frontiers and Innovations, v. 6, n. 2, pp. 86–101.
- [99] PLESSIX, R.-E., 2006, “A review of the adjoint-state method for computing the gradient of a functional with geophysical applications”, Geophysical Journal International, v. 167, n. 2, pp. 495–503.
- [100] QAWASMEH, A., HUGUES, M. R., CALANDRA, H., et al., 2017, “Performance portability in reverse time migration and seismic modelling via OpenACC”, The International Journal of High Performance Computing Applications, v. 31, n. 5, pp. 422–440.
- [101] QU, L., ABDELKHALAK, R., LTAIEF, H., et al., 2022, “Exploiting temporal data reuse and asynchrony in the reverse time migration”, The International Journal of High Performance Computing Applications, p. 10943420221128529.
- [102] RODEN, J. A., GEDNEY, S. D., 2000, “Convolution PML (CPML): An efficient FDTD implementation of the CFS–PML for arbitrary media”, Microwave and optical technology letters, v. 27, n. 5, pp. 334–339.
- [103] SCHUSTER, G. T., 2017, Seismic inversion. Tulsa, OK U.S.A. 74137-3575, Society of Exploration Geophysicists.
- [104] SERPA, M. S., CRUZ, E. H., DIENER, M., et al., 2019, “Optimization strategies for geophysics models on manycore systems”, The International Journal of High Performance Computing Applications, v. 33, n. 3, pp. 473–486.
- [105] SERPA, M. S., PAVAN, P. J., CRUZ, E. H., et al., 2021, “Energy efficiency and portability of oil and gas simulations on multicore and graphics processing unit architectures”, Concurrency and Computation: Practice and Experience, p. e6212.
- [106] SIAHKOORI, A., RIZZUTI, G., HERRMANN, F. J., 2020, “Uncertainty quantification in imaging and automatic horizon tracking—a Bayesian deep-prior based approach”. In: SEG Technical Program Expanded Abstracts 2020, Society of Exploration Geophysicists, pp. 1636–1640.
- [107] SILVA, B. D. S., SOARES FILHO, D. M., LANDAU, L., 2021, “Full waveform inversion of areal shot records”, Journal of Applied Geophysics, v. 193, pp. 104427.
- [108] SILVA, B., SILVA, J., LANDAU, L., 2015, “Optimized finite differences scheme applied to the acoustic wave equation”. In: 14th International Congress of

the Brazilian Geophysical Society & EXPOGEF, Rio de Janeiro, Brazil, 3-6 August 2015, pp. 1100–1105. Brazilian Geophysical Society, August.

- [109] SILVA, K. C., 2012, Modelagem, MRT e Estudos de Iluminação empregando o conceito de dados sísmicos blended. Tese de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Janeiro. Disponível em: <http://www.coc.ufrj.br/pt/dissertacoes-de-mestrado/112-msc-pt-2012/2265-karen-carrilho-da-silva>.
- [110] SOMMER, V. P., KUCHLE, J., DE ROS, L. F., 2022, “Seismic stratigraphic framework and seismic facies of the Aptian Pre-salt Barra Velha Formation in the Tupi Field, Santos Basin, Brazil”, Journal of South American Earth Sciences, v. 118, pp. 103947.
- [111] STOLTZFUS, L., HAGEDORN, B., STEUWER, M., et al., 2019, “Tiling optimizations for stencil computations using rewrite rules in lift”, ACM Transactions on Architecture and Code Optimization (TACO), v. 16, n. 4, pp. 1–25.
- [112] SUN, W., FU, L.-Y., 2013, “Two effective approaches to reduce data storage in reverse time migration”, Computers & Geosciences, v. 56, pp. 69–75.
- [113] SYMES, W. W., 2007, “Reverse time migration with optimal checkpointing”, Geophysics, v. 72, n. 5, pp. SM213–SM221.
- [114] TARANTOLA, A., 1984, “Inversion of seismic reflection data in the acoustic approximation”, Geophysics, v. 49, n. 8, pp. 1259–1266.
- [115] TARANTOLA, A., 1986, “A strategy for nonlinear elastic inversion of seismic reflection data”, Geophysics, v. 51, n. 10, pp. 1893–1903.
- [116] WANG, W., ZHANG, W., LEI, T., 2023, “Compression of seismic forward modeling wavefield using TuckerMPI”, Computers & Geosciences, p. 105298.
- [117] WANG, Y., 2015, “Frequencies of the Ricker wavelet”, Geophysics, v. 80, n. 2, pp. A31–A37.
- [118] WANG, Y., GU, Y., LIU, J., 2020, “A domain-decomposition generalized finite difference method for stress analysis in three-dimensional composite materials”, Applied Mathematics Letters, v. 104, pp. 106226.
- [119] WANG, Z.-Y., HUANG, J.-P., LIU, D.-J., et al., 2019, “3D variable-grid full-waveform inversion on GPU”, Petroleum Science, v. 16, n. 5, pp. 1001–1014.

- [120] WEI, Z., MEI, J., WU, Z., et al., 2021, “FWI imaging: Revealing the unprecedented resolution of seismic data”. In: SEG Technical Program Expanded Abstracts. SEG, September.
- [121] WILEY, R., MARFURT, K. J., 2006, “Marmousi2: An elastic upgrade for Marmousi”, The Leading Edge, v. 25, pp. 156–166.
- [122] WOLF, M., 2014, High-performance embedded computing: applications in cyber-physical systems and mobile computing. Burlington, MA, USA, Morgan Kaufmann; 2nd edition.
- [123] WOLF, M. E., LAM, M. S., 1991, “A loop transformation theory and an algorithm to maximize parallelism”, IEEE Transactions on Parallel & Distributed Systems, v. 2, n. 04, pp. 452–471.
- [124] YAO, G., WU, D., WANG, S.-X., 2020, “A review on reflection-waveform inversion”, Petroleum Science, v. 17, n. 2, pp. 334–351.
- [125] YILMAZ, Ö., 2001, Seismic data analysis: Processing, inversion, and interpretation of seismic data. Society of exploration geophysicists.
- [126] YOKOKAWA, M., NAKAI, A., KOMATSU, K., et al., 2020, “I/o performance of the sx-aurora tsubasa”. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 27–35. IEEE, May.
- [127] ZAND, T., MALCOLM, A., GHOLAMI, A., et al., 2019, “Compressed imaging to reduce storage in adjoint-state calculations”, IEEE Transactions on Geoscience and Remote Sensing, v. 57, n. 11, pp. 9236–9241.
- [128] ZHANG, N., DRISCOLL, M., MARKLEY, C., et al., 2017, “Snowflake: A lightweight portable stencil dsl”. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 795–804. IEEE, July.
- [129] ZHANG, X., CURTIS, A., 2020, “Variational full-waveform inversion”, Geophysical Journal International, v. 222, n. 1, pp. 406–411.
- [130] ZHANG, Z., WU, Z., WEI, Z., et al., 2020, “FWI Imaging: Full-wavefield imaging through full-waveform inversion”. In: SEG International Exposition and Annual Meeting. OnePetro, October.
- [131] ZHAO, K., DI, S., LIAN, X., et al., 2020, “SDRBench: Scientific data reduction benchmark for lossy compressors”. In: 2020 IEEE International Conference on Big Data (Big Data), pp. 2716–2724. IEEE, March.

- [132] ZHAO, Z., SEN, M. K., 2019, “A gradient based MCMC method for FWI and uncertainty analysis”. In: SEG Technical Program Expanded Abstracts 2019, Society of Exploration Geophysicists, pp. 1465–1469.
- [133] ZHAO, Z., SEN, M. K., 2021, “A gradient-based Markov chain Monte Carlo method for full-waveform inversion and uncertainty analysis”, Geophysics, v. 86, n. 1, pp. R15–R30.
- [134] ZHAO, Z., SEN, M. K., 2022, “Uncertainty quantification for full-waveform inversion with a mini-batch gradient-based sampling method”. In: Second International Meeting for Applied Geoscience & Energy, pp. 897–901. Society of Exploration Geophysicists and the American Association of Petroleum Geologists, August.
- [135] ZHOU, H.-W., HU, H., ZOU, Z., et al., 2018, “Reverse time migration: A prospect of seismic imaging methodology”, Earth-science reviews, v. 179, pp. 207–227.
- [136] ZHU, H., LI, S., FOMEL, S., et al., 2016, “A Bayesian approach to estimate uncertainty for full-waveform inversion using a priori information from depth migration”, Geophysics, v. 81, n. 5, pp. R307–R323.

# Appendices

# Appendix A

## Boundary Conditions

In this Appendix, we present three of the main boundary conditions used in seismic imaging applications, such as wavefield propagation, RTM, LS-RTM, and FWI, among others. The first two boundary conditions discussed in sections A.1 and A.2 are classified as absorbing boundary conditions (ABCs). The ABCs aim to attenuate the seismic signals on artificial boundaries simulating infinite domains. Lastly, we present in section A.3 the random boundary condition proposed by CLAPP [32], which explores the non-coherent nature of the seismic amplitudes on the artificial boundaries to build imaging conditions of migration/inversion methods.

### A.1 Non-reflecting boundary condition

One of the most simple and used non-reflecting boundary conditions was proposed by CERJAN *et al.* [25]. The Cerjan sponge boundary consists of applying a Gaussian function to the wave equation solution through time. However, it is necessary to extend the domain  $\Omega \subset \mathbb{R}^{n_{sd}}$ ,  $n_{sd} = 2, 3$  where the solution  $p(\mathbf{r}, t)$  is defined. Thus, the pressure  $p$ , and the slowness  $\sigma$  in the wave equation 2.2 can be redefined in the closed domain  $\bar{\Omega} = \Omega \cup \Gamma \subset \mathbb{R}^{n_{sd}}$ , where  $\Omega \cap \Gamma = \emptyset$ . Thus, we define the subset  $\Gamma$  as the one where the wavefield amplitudes are attenuated by the Gaussian function. In this context, the attenuation function can be formulated as:

$$g(\xi) = \begin{cases} 1, & \text{if } \xi \in \Omega \\ e^{-[\lambda(N-\xi)]^2}, & \text{if } \xi \in \Gamma \end{cases} \quad (\text{A.1})$$

where  $N$  is a positive real number,  $\xi \in \{x, y, z\}$ , and  $\lambda$  is the damping factor which controls wavefield attenuation.

The discretization of Gaussian equation A.1 by FD produces the damping array  $g_i = \exp\{-[\lambda(N-i)]^2\}$ , where  $i \in \{x, y, z\}$ , and  $N$  is the size of the region where the wavefield amplitude is attenuated (size thickness). The values of  $g_i$  are applied to the

wavefield in the region related to the subset  $\Gamma$ , that is the region where the damping array attenuates the wavefield amplitudes provided by the numerical solution of the wave equation. Through numerical experiments, CERJAN *et al.* [25] provided empirical values for the parameters, where  $N = 20$  and  $\lambda = 0.015$ . However, BORDING [18] pointed out that these parameters are not optimized, suggesting  $N = 45$  and  $\lambda = 0.0053$ .

## A.2 Convolutional perfectly matched layer

The Convolutional Perfectly Matched Layer (CPML) equations are obtained by introducing a complex coordinate stretching parameter which is applied to the wave equations in the frequency domain. After that, we use the recursive convolution algorithm to convert the equations to the time-domain [63, 97]. Thus, the partial derivative in the stretched coordinate space after applying the recursive convolution method [102] is given by:

$$\frac{\partial}{\partial \tilde{i}} = \frac{\partial}{\partial i} + \psi_i. \quad (\text{A.2})$$

where  $i \in \{x, y, z\}$ ,  $\tilde{i}$  is known as the stretched coordinate, and  $\psi_i$  is an auxiliary variable whose time evolution is defined as:

$$\psi_i^n = a_i \psi_i^{n-1} + b_i \left( \frac{\partial}{\partial i} \right), \quad (\text{A.3})$$

with  $n$  representing the temporal evolution.

The parameters  $a_i$  and  $b_i$  follow the expressions:

$$a_i = e^{-(\sigma_i + \alpha_i)\Delta t}; \quad b_i = \frac{\sigma_i}{\sigma_i + \alpha_i} (a_i - 1). \quad (\text{A.4})$$

where  $\sigma_i$  is a damping factor that controls the absorption of the wavefield amplitudes.  $\alpha_i$  is a real positive parameter that improves absorption of low-frequency components [97].

### A.2.1 Application to isotropic acoustic wave equation

Remembering that the acoustic isotropic wave equation is:

$$\frac{\partial p^2(\mathbf{r}, t)}{\partial x^2} + \frac{\partial p^2(\mathbf{r}, t)}{\partial y^2} + \frac{\partial p^2(\mathbf{r}, t)}{\partial z^2} - \sigma(\mathbf{r})^2 \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} = f(\mathbf{r}_s, t), \quad (\text{A.5})$$

the second spatial derivatives can be expressed in terms of the first derivatives as follows:

$$\frac{\partial^2 p(\mathbf{r}, t)}{\partial i^2} = \frac{\partial}{\partial i} \left( \frac{\partial p(\mathbf{r}, t)}{\partial i} \right). \quad (\text{A.6})$$

Using the formulation of equation A.6, the wave equation A.5 can be written in the

stretched coordinate space as:

$$\frac{\partial p_x(\mathbf{r},t)}{\partial x} + \frac{\partial p_y(\mathbf{r},t)}{\partial y} + \frac{\partial p_z(\mathbf{r},t)}{\partial z} - \sigma(\mathbf{r})^2 \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} = f(\mathbf{r}_s,t), \quad (\text{A.7})$$

where  $p_i(\mathbf{r},t)$  is defined as:

$$p_i(\mathbf{r},t) = \frac{\partial p_i(\mathbf{r},t)}{\partial \tilde{t}}. \quad (\text{A.8})$$

Applying equation A.2 to equation A.8,  $p_i$  can be expressed as:

$$p_i(\mathbf{r},t) = \frac{\partial p(\mathbf{r},t)}{\partial i} + \psi_i(\mathbf{r}). \quad (\text{A.9})$$

where the auxiliary variable  $\psi_i^n(\mathbf{r})$  follows the temporal evolution:

$$\psi_i^n(\mathbf{r}) = a_i \psi_i^{n-1}(\mathbf{r}) + b_i \left( \frac{\partial p(\mathbf{r},t)}{\partial i} \right). \quad (\text{A.10})$$

Inserting equations A.2 and A.9 into equation A.7 produces the isotropic acoustic wave equation in the stretched coordinate space:

$$\begin{aligned} \sigma(\mathbf{r})^2 \frac{\partial^2 p(\mathbf{r},t)}{\partial t^2} = & \nabla^2 p(\mathbf{r},t) \\ & + \frac{\partial \psi_x(\mathbf{r})}{\partial x} + \frac{\partial \psi_y(\mathbf{r})}{\partial y} + \frac{\partial \psi_z(\mathbf{r})}{\partial z} \\ & + \zeta_x(\mathbf{r}) + \zeta_y(\mathbf{r}) + \zeta_z(\mathbf{r}) \\ & - f(\mathbf{r}_s,t), \end{aligned} \quad (\text{A.11})$$

where  $\zeta_i(\mathbf{r})$  are auxiliary variables with temporal evolution given by:

$$\zeta_i^n(\mathbf{r}) = a_i \zeta_i^{n-1}(\mathbf{r}) + b_i \left[ \left( \frac{\partial^2 p(\mathbf{r})}{\partial i^2} \right)^n + \left( \frac{\partial^2 \psi_i(\mathbf{r})}{\partial i^2} \right)^n \right]. \quad (\text{A.12})$$

As better explained in section A.1,  $p(\mathbf{r},t)$  and  $\sigma(\mathbf{r})$  have to be defined in the closed domain  $\bar{\Omega} = \Omega \cup \Gamma \subset \mathbb{R}^{n_{sd}}$ . But,  $\zeta_i(\mathbf{r})$  and  $\psi_i(\mathbf{r})$  are defined in  $\Gamma$ . Thus, we solve the wave equation 2.2 or A.5 in  $\Omega$ , and the CPML equation A.11 in  $\Gamma$ .

The partial derivatives of equation A.11 can be discretized by the FDM as the wave equation in section 3.1. The FD discretization of equation A.11 produces the vectors  $\sigma$ ,  $\mathbf{p}$ ,  $\boldsymbol{\psi}$ , and  $\boldsymbol{\zeta}$ , and its solution is the wavefield attenuation on the discretized region related to the artificial boundary  $\Gamma$ .

### A.3 Random boundary condition

CLAPP [32] proposed create an artificial boundary beyond the domain  $\Omega \subset \mathbb{R}^{n_{sd}}$  by assigning random velocities to it. The idea is based on non-coherent reflections coming from the boundaries that have minimum impact on the calculation of the RTM imaging condition. Thus, instead of suppressing unwanted waves by inserting new equations into the problem as shown in section A.1 and A.2, random boundaries are based on exploring low correlations with non-coherent signals coming from an artificial boundary with random velocities.

Following what we introduce in sections A.1 and A.2,  $\bar{\Omega} = \Omega \cup \Gamma \subset \mathbb{R}^{n_{sd}}$  is the closed domain where the wave equation solution  $p(\mathbf{r}, t)$  is defined. The slowness field is also defined in  $\bar{\Omega}$  and follows the expression:

$$\sigma(\mathbf{r}) = \begin{cases} 1/v(\mathbf{r}), & \text{if } \mathbf{r} \in \Omega \\ 1/(v(\mathbf{r}) + r * \mathbf{r}), & \text{if } \mathbf{r} \in \Gamma \end{cases} \quad (\text{A.13})$$

where  $r$  is a random number that assumes values following the uniform distribution  $U[0, 1]$ . According to CLAPP [31] the values of  $v(\mathbf{r}) + r * \mathbf{r}$  have to pass through a stability test before solving the wave equation with numerical methods. We use the FDM to discretize equation A.13, and the vector  $\sigma$  has the velocity values of the medium and random velocity on the region related to the artificial boundary  $\Gamma$ .

# Appendix B

## Wavefield Decimation

Emerging techniques to compress huge volumes of data became essential, mainly for scientific applications, due to the widening gap between computer speed and I/O [4, 66, 76]. One of the simplest data compression techniques is known as decimation, which consists of retaining every  $s$ th datum (where  $s = 10$  generally by default, as suggests its name) and discarding the remainder. The datum, in the simplest case, can be a scalar or, in more complex scenarios, an array containing every state variable at all points of a spatial grid [4]. The data reduction produced by the factor  $s$  classifies the decimation as a lossy compression technique.

In this sense, let  $\{u_i\}_{i=0}^N$  be a data produced by computational simulations of some physical system. Decimating  $\{u_i\}_{i=0}^N$  transforms it to the data  $\{w_j\}_{j=0}^{N/s}$ , where  $s = 10$ , and  $N$  is a number divisible by  $s$ . The decimation process is described in Algorithm 12. The algorithm is quite simple consisting of retaining the  $s$ th datum (lines 3 and 4) and discarding everything else.

---

**Algorithm 12** Decimation

---

**Require:** uncompressed data  $\mathbf{u}$ , and stride  $s$

```
1: function DECIMATE (  $\mathbf{u}$ ,  $s$  )
2:   for  $j = 0$  to  $N/s$  do
3:      $i = j * s$ 
4:      $w_j \leftarrow u_i$ 
5:   end for
6:   store decimated data  $w_j^{N/s}$ 
7: end function
```

---

Considering the computational implementation of the isotropic acoustic wave equation, we can use Algorithm 12 to decrease the amount of information to be stored in the disk concerning the wavefield (wave equation solution). However, it is necessary to verify if the decimation factor respects the Nyquist theory, which depends on the highest and lowest frequency of the seismic source signal [112]. Recalling that  $\Delta t$  is the time step for FD discretization of the wave acoustic wave equation, and  $\Delta t_{nyq}$  the Nyquist time step

(see equation 3.18), the decimation factor is defined as:

$$s \leq \frac{\Delta t_{nyq}}{\Delta t} = \frac{1}{2\Delta t(f_{max} - f_{min})}, \quad (\text{B.1})$$

Equation B.1 imposes an upper limit to the value of  $s$ , and when required such value must be less than what the decimation proposes in the first place ( $s = 10$ ). On the other hand, the decimation factor can reach higher values depending on the highest and lowest seismic source frequency. Hence, Algorithm 13 shows the decimation of the wavefield solution for the isotropic acoustic wave equation. In this case, the temporal evolution of the wavefield is decimated by the factor of  $s$  when the variable  $n$  meets it (lines 9 to 12). Lines 5 to 7 show the wave equation discretization that provides the wavefield. Lines 13 and 14 update the solutions over time, and line 16 stores the decimated wavefield. Algorithm 13 can be used to decrease the amount of wavefield data to be stored in disk, which is required by the simplest way to implement the RTM (see Algorithm 2). Besides, we implement it to store the ZFP compressed wavefield in Algorithm 3 (line 10).

---

**Algorithm 13** Wavefield decimation

---

**Require:** auxiliary matrix  $\mathbf{C}$ , FD coefficients  $\mathbf{b}$ , and stride  $s$

```

1: function DECIMATE_WAVEFIELD (  $\mathbf{C}$ ,  $\mathbf{b}$ ,  $s$  )
2:   apply initial conditions
3:   initialize  $l = 0$ 
4:   for  $n = 0$  to  $N_t$  do
5:      $P_{i,j,k}^{n+1} = C_{i,j,k}[2b_0 +$ 
6:        $\sum_{m=1}^1 b_m (P_{i+m,j,k}^n + P_{i-m,j,k}^n + P_{i,j+m,k}^n + P_{i,j-m,k}^n + P_{i,j,k+m}^n + P_{i,j,k-m}^n)]$ 
7:      $-2P_{i,j,k}^n + P_{i,j,k}^{n-1} - f_{i,j,k}^n$ 
8:     apply boundary conditions
9:     if  $\text{mod}(n, s) = 0$  then
10:       $w_{i,j,k}^l \leftarrow P_{i,j,k}^n$ 
11:       $l = l + 1$ 
12:     end if
13:      $P_{i,j,k}^{n-1} \leftarrow P_{i,j,k}^n$ 
14:      $P_{i,j,k}^n \leftarrow P_{i,j,k}^{n+1}$ 
15:   end for
16:   store decimated data  $w_{i,j,k}^{N_t/s}$ 
17: end function

```

---

## Appendix C

# Source and Reconstructed Wavefield Equivalence

Let the acoustic wave equation be given by,

$$\begin{aligned} \nabla^2 p(\mathbf{r}, t) - \frac{1}{v^2(\mathbf{r})} \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} &= f(t) \delta(\mathbf{r}_s - \mathbf{r}), \\ p(\mathbf{r}, t) &= 0 \quad \text{on } \partial\Omega \\ p(\mathbf{r}, 0) = 0 \quad \text{and} \quad \frac{\partial p(\mathbf{r}, 0)}{\partial t} &= 0, \quad \mathbf{r} \in \Omega, \end{aligned} \tag{C.1}$$

where  $t \in [0, T]$ , and the reconstruction acoustic wave equation, such that

$$\begin{aligned} \nabla^2 p^R(\mathbf{r}, \tau) - \frac{1}{v^2(\mathbf{r})} \frac{\partial^2 p^R(\mathbf{r}, \tau)}{\partial \tau^2} &= f(\tau) \delta(\mathbf{r}_s - \mathbf{r}), \\ p^R(\mathbf{r}, \tau) &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{C.2}$$

with  $\tau \in [0, T]$  with  $\tau = T - t$ .

Manipulating equations [C.1](#) and [C.2](#):

$$\begin{aligned} \nabla^2 p(\mathbf{r}, t) - \frac{1}{v^2(\mathbf{r})} \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} - f(t) \delta(\mathbf{r}_s - \mathbf{r}) &= \nabla^2 p^R(\mathbf{r}, \tau) - \frac{1}{v^2(\mathbf{r})} \frac{\partial^2 p^R(\mathbf{r}, \tau)}{\partial \tau^2} - f(\tau) \delta(\mathbf{r}_s - \mathbf{r}), \\ \nabla^2 p(\mathbf{r}, t) - \frac{1}{v^2(\mathbf{r})} \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} - f(t) \delta(\mathbf{r}_s - \mathbf{r}) - \nabla^2 p^R(\mathbf{r}, \tau) + \frac{1}{v^2(\mathbf{r})} \frac{\partial^2 p^R(\mathbf{r}, \tau)}{\partial \tau^2} + f(\tau) \delta(\mathbf{r}_s - \mathbf{r}) &= 0, \\ \nabla^2 [p(\mathbf{r}, t) - p^R(\mathbf{r}, \tau)] - \frac{1}{v^2(\mathbf{r})} \left[ \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} - \frac{\partial^2 p^R(\mathbf{r}, \tau)}{\partial \tau^2} \right] - [f(t) - f(\tau)] \delta(\mathbf{r}_s - \mathbf{r}) &= 0, \end{aligned}$$

We can verify that the wavefields are equivalent if:

$$\begin{aligned} p(\mathbf{r}, t) - p^R(\mathbf{r}, \tau) &= 0 \\ \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} - \frac{\partial^2 p^R(\mathbf{r}, \tau)}{\partial \tau^2} &= 0 \\ f(t)\delta(\mathbf{r}_s - \mathbf{r}) - f(\tau)\delta(\mathbf{r}_s - \mathbf{r}) &= 0, \end{aligned}$$

for  $\tau = 0, t = T$ . Then, the necessary initial conditions for equation C.2 to reconstruct the source wavefield are given by,

$$\begin{aligned} p^R(\mathbf{r}, 0) &= p(\mathbf{r}, T) \\ \frac{\partial^2 p^R(\mathbf{r}, 0)}{\partial \tau^2} &= \frac{\partial^2 p(\mathbf{r}, T)}{\partial t^2} \\ f(0)\delta(\mathbf{r}_s - \mathbf{r}) &= f(T)\delta(\mathbf{r}_s - \mathbf{r}) \end{aligned}$$

# Appendix D

## Vector Processing

The vector processing consists of simultaneously operating on different elements of one-dimensional arrays (vectors) rather than single ones [24, 55]. This approach is classified as data-level parallelism (DLP) to explore parallelism in nowadays computer architectures. In addition to DLP, we have different types of parallelism at the instruction level, multi-threading, core, socket, and node [1]. A briefly description of them according to BUCHTY et al. [21] is listed bellow:

1. Instruction-level parallelism: exploits parallel processing of instructions at runtime by collecting a set of data-flow-driven techniques,
2. Data-level parallelism (DLP): apply an instruction to all elements of an array at once (or in parallel),
3. Multi-threading parallelism: consists of time-sharing the processor resources, where one thread can execute instructions from another one,
4. Core-level parallelism: contains several cores of the same characteristics grouped together to build a multicore processor strongly coupled,
5. Socket-level parallelism: groups of cores are attached to compose a socket. It is also possible for GPUs by connecting multi-GPUs via PCI Express (PCIe),
6. Node-level parallelism: in this level one node contains groups of multicore/GPUs connected to hundreds of others nodes with the same technologies on a specific topology.

Restricting ourselves to the DLP, vector processing has been present since the 1970s, where the emerging vector architectures exploited data parallelism with long vectors of thousands of bits. Nowadays, it is more common to find hardware focused on short-vector (up to 2048 bits) designs based on the widespread vector implementation known as the

single-instruction-multiple-data extension (SIMD). Be aware that SIMD instructions depend on the processor's register size [24]. For instance, the Streaming SIMD Extensions (SSE) developed in the late 90's for the Intel company had registers of 128 bits which holds 2 double-precision floats or 4 single-precision floats. In 2010, the registers' size doubled to 256 bits for floating point operations, but it was not expanded for the integer SIMD instructions. Such implementation became known as Advanced Vector Extensions (AVX). After that, the AVX2 extensions were introduced in the Haswell processor (2013) expanding the 128-bit SIMD instructions to 256 bits and introducing fused multiply-add instructions (FMA). Finally, the most recent AVX is the AVX-512 with registers of 512 bits for floating point and integer operations. We can find such implementations in the Intel Xeon Phi and Skylake CPUs (2015) [21, 55].

## D.1 Basic architecture of NEC SX-Aurora TSUBASA

Vector processing developments are not restricted to the Intel company. Other companies such as ARM, IBM, and Fujitsu have released their SIMD instruction technologies. Opposite to short-vector designs, the NEC company is one of the exceptions that developed dedicated vector systems with registers that reach 16,384 bits [24]. Different from traditional vector engines, the newest SX vector supercomputer with a dedicated vector processor from NEC company, known as SX-Aurora TSUBASA, consists of a vector host (VH) and one or more vector engines (VEs). The VH is an x86 processor that handles system calls from applications running on the VE. The VE, on the other hand, is a vector processor responsible for executing the entire application. Besides, it contains eight vector cores physically implemented as a PCIe with three components which are a scalar processing unit (SPU), vector processing unit (VPU), and memory system. The computations are made mostly for the VPUs, while the SPUs provide functionalities similar to traditional CPUs [42, 64]. Furthermore, each vector core on the VPUs has three vectors fused multiply-add (VFMA) units which independently operate by different vector instructions, and one vector instruction executes 256 arithmetic operations with eight clock cycles. The memory subsystem has six advanced high bandwidth memory (HBM) modules integrated into the CPU package of a VE [42]. Depending on the SX-series, the clock frequency of each vector core and memory bandwidth per VE are different. The type 10B series has a clock frequency of 1.4 GHz and a memory bandwidth of 1.22 TB/s. For the latest series, the type 20B, the clock frequency is 1.6 GHz and the memory bandwidth is 1.53 TB/s [42, 64].

Although the hardware configuration of SX-Aurora TSUBASA looks similar to conventional heterogeneous systems, such as those based on GPU systems, the design concept is different because the entire application executes on the VE. This concept brings two major advantages, which are negligible data transfer among the VH and VEs for many

cases and no special programming languages. Thus, a program coded in standard C/C++ or FORTRAN language, for instance, can be compiled and run as it is [42]. In this sense, the SX-Aurora TSUBASA provides automatic vectorization and parallelization such as other computational optimizations that can be enabled by compiler flags. Here, we describe the basic ones we use for seismic modeling and RTM applications. According to NEC SX- Aurora TSUBASA guide<sup>1</sup> the following flags are defined:

1. `-mparallel`: allows automatic parallelization. `-pthread` is implicitly enabled,
2. `-mparallel-innerloop`: allows to parallelize inner-loops,
3. `-fivdep`: inserts `ivdep` directive before all loops, where `ivdep` regards the unknown dependency as vectorizable dependency during the automatic vectorization.

Aiming to verify the program execution information about the computational optimizations based on the listed compilation flags, we use the performance analysis tools<sup>2</sup> `proginf` and `ftrace`. Hence, `proginf` outputs the program execution analysis information using SX-Aurora performance counter, that is, it outputs items related to vector instructions such as elapsed time, vector instruction execution time, the ratio of vector operations to all operations, cache missing time, peak usage memory, among others. The second tool, `ftrace`, is used to obtain performance information such as CPU usage and vectorization aspects of each function in a program as well as program regions. Further details can be found in the `proginf` and `ftrace` manual (see footnote #2).

---

<sup>1</sup><https://sxaoratsubasa.sakura.ne.jp/documents/sdk/pdfs/g2af01e-C++UsersGuide-029.pdf>

<sup>2</sup>[https://sxaoratsubasa.sakura.ne.jp/documents/sdk/pdfs/g2at03e-PROGINF\\_FTRACE\\_User\\_Guide\\_en.pdf](https://sxaoratsubasa.sakura.ne.jp/documents/sdk/pdfs/g2at03e-PROGINF_FTRACE_User_Guide_en.pdf)

# Appendix E

## The basics of OpenACC

According to the OpenACC specification<sup>1</sup>, OpenACC is a collection of compiler directives, library routines and environment variables using so-called directives to express parallelism inherent in your code. Because of that, OpenACC is classified as a high-level, directive-based programming model which augments a given code by inserting hints into the C/C++ and FORTRAN programs aiming to achieve parallelism [26].

Currently, the OpenACC compilers can target the following hardware platforms:

1. traditional X86 systems;
2. traditional multicore platforms;
3. accelerators such as GPUs;
4. OpenPOWER, and Xeon Phi processors;
5. Knights Landing (KNL), and advanced RISC machines (ARM) processors.

OpenACC directives are set up in the following structure (for C/C++<sup>2</sup>):

```
#pragma acc <directive> [clause [[,] clause] ... newline]
```

The keyword `#pragma` is a compiler directive used in C/C++ programming, and it is followed by the OpenACC directive `acc`. To activate this directive, we have to include the OpenACC header statement. After that, it comes directives that tell the compiler what to do. Further instructions can be obtained by adding one or more optional clauses. The present statement form is known as OpenACC construct.

OpenACC construct provides three directive types and clause categories which tell something for the compiler to do about the code block that follows it. The directive types are related to computing, data management, and synchronization. There are also three

---

<sup>1</sup>[https://www.openacc.org/sites/default/files/inline-files/OpenACC\\_2pt5.pdf](https://www.openacc.org/sites/default/files/inline-files/OpenACC_2pt5.pdf)

<sup>2</sup>We omit the guidelines for the FORTRAN language aiming to simplify the theory about OpenACC.

clause categories, related to data handling, work distribution, and control flow. Below, we present all the directives and clauses for each class discussed earlier:

1. Compute directives: `parallel`, `kernels`, `routine`, and `loop`;
2. Data management directives: `data`, `update`, `cache`, `atomic`, `declare`, `enter data`, and `exit data`;
3. Synchronization directive: `wait`;
4. Data handling clauses: `default`, `private`, `firstprivate`, `copy`, `copyin`, `copyout`, `create`, `delete`, and `deviceptr`;
5. Work distribution clauses: `seq`, `auto`, `gang`, `worker`, `vector`, `tile`, `num_gangs`, `num_workers`, and `vector_length`;
6. Control flow: `if`, `if_present`, `independent`, `reduction`, `async`, and `wait`.

Aiming at not being extensive in detailing each directive and clause, we present in the next sections the main ones that we use to implement the RTM. The whole set of directives reduces to the `kernels`, `loop`, `data`, and `update` directives, while the clauses reduce to the `create`, `copy`, `copyin`, `copyout`, and `independent` clauses. More details about all directives and clauses can be seen in the OpenACC specification<sup>3 4 5</sup>.

## E.1 Compute directives

**Kernels** the `kernels` compute construct offloads a region of code to the accelerator device that may contain parallelism. In this case, the compiler is responsible to analyze and decide what to parallelize and how to distribute the work on the computing hardware. Thus, the `kernels` constructs rely on the automatic parallelization capabilities of the compiler to analyze the region and identify safety parallelism.

**Loop** the `loop` directive extends the `kernels` construct by telling the compiler that a loop is independent when it is not possible to determine at compile time.

## E.2 Data management directives

**Data** the `data` directive is a structured construct that facilitates sharing data among multiple parallel regions. It must begin and end in the same scope, such as the same function or subroutine.

---

<sup>3</sup><https://www.openacc.org/>

<sup>4</sup><https://www.openacc.org/sites/default/files/inline-files/openacc-guide.pdf>

<sup>5</sup><https://www.openacc.org/sites/default/files/inline-files/API%20Guide%202.7.pdf>

**Update** the `update` directive allows keeping both host and device content synchronized on their memories by copying the values from each other. This can be done using the clauses `self` and `device`. The first one copies the data from memory device to the host memory. On the other hand, the `device` updates the memory device with the data on the host.

### E.3 Data handling clauses

**Create** the `create` clause is the foundation of the clauses that allocate data. It is responsible for creating a data lifetime for the objects listed in the clause on the device. Thus, if the objects have had memory created for them on the device, then the runtime only updates them on the device when an object's memory is no longer needed. However, if the objects do not have memory assigned to it, then the runtime allocates memory for them and updates the reference count.

**Copy** The `copy` clause starts with a `present`<sup>6</sup> clause which is the first part of every data clause. Thus, if the data is not present, the `create` clause is then performed. After that, the `copy` clause copies the data from the host to the device at the beginning of the region, and when the region ends the data is copied back to the host and the memory is freed.

**Copyin** the `copyin` clause does what the `copy` clause does on the way into a data region, except at the end of the region when the data is not copied back to the host.

**Copyout** finally, the `copyout` clause does everything the `copy` clause does, but it does not copy the data on the data region entry.

### E.4 Control flow clauses

**Independent** the `independent` clause can be only applied to loops in kernels. This is useful because in some scenarios the compiler does not have enough information to decide if a code region (kernel regions) can be safely parallelized or not. Thus, the `independent` clause tells the compiler the kernel region can be parallelized without any problem. If not, the program may lose performance due to the kernel region being run sequentially.

---

<sup>6</sup>According to OpenACC guide, the `present` clause is more a bookkeeping clause than a data clause which ensures that the objects are already on the device so that the computation can be sent to the device.

# Appendix F

## Workflow main building blocks

The main building blocks of the three sequential stages in Algorithm 9 are (i) Bayesian tomography to estimate the seismic velocity from first arrival travel times, where the forward modeling relies on the eikonal equation (called here Bayesian Eikonal Tomography - BET), and the Reversible Jump Monte Carlo Markov Chain (RJMCMC) algorithm [12, 53, 54]; (ii) Seismic migration using Reverse Time Migration (RTM) wrapped in a Monte Carlo algorithm to propagate the uncertainties; (iii) Image interpretation supported by statistical information, post-processing the ensemble of seismic images. The BET inversion for velocity estimation offers the flexibility of treating any prior information, like a reasonable initial velocity estimation, and it has the advantage of providing results that help to quantify the uncertainties associated with the solutions [12]. RTM migrates the seismograms for the velocity models from the first stage (BET) to generate the corresponding set of migrated seismic images. Lastly, uncertainty maps give a global representation of the ensemble of velocity fields and images produced respectively by BET and RTM.

### F.1 Estimating uncertain velocity fields

The raw data for the workflow and also of its first stage, as made explicit in Algorithm 9, includes seismograms measured through sensors placed near the surface, the domain to be imaged, and source signatures. They all are noisy or inaccurate to a certain degree, what entails, alongside with data scarcity, uncertainties in the velocity field estimation, the main goal of this first stage. Here, such an estimation is carried out by tomography [53]. Seismic tomography is an inversion tool that aims to identify parameters characterizing the physics of waves propagating within the subsurface (e.g., velocity, anisotropy, among others) given a set of observations (measurement data). The following observation equation relates parameters and the data,

$$\mathbf{d}^{obs} = \mathbf{F}(\mathbf{m}) + \boldsymbol{\varepsilon}, \quad (\text{F.1})$$

where  $\varepsilon$  is an additive noise encompassing model and measurements errors.

The high-dimensional vector  $\mathbf{m} = (\mathbf{V}, \mathbf{U}, n)$  is parametrized by a set of Voronoi polygons. The components of  $\mathbf{m}$  are the number of polygons (or cells)  $n$ , a  $n$ -dimensional vector containing the cell nucleus positions  $\mathbf{U} = \{u_i\}$ , and the velocity of the compressional wave  $\mathbf{V} = \{v_i\}$  assigned to each cell. As described in BODIN and SAMBRIDGE [16], the Voronoi polygons are non-overlapping regions  $R_1, R_2, \dots, R_n$  where the points within  $R_i$  are closer to  $u_i$  than any of the other  $u_j (i \neq j)$ . The forward operator  $\mathbf{F}$  encapsulates the simplified physics resulting from assuming high-frequency wave propagation and ray tracing. Therefore,  $\mathbf{F}$  is a nonlinear mapping between the velocity field and the first arrival travel times captured by the sensors [20], using the eikonal equation,

$$\left(\frac{\partial T(\mathbf{r})}{\partial x}\right)^2 + \left(\frac{\partial T(\mathbf{r})}{\partial y}\right)^2 + \left(\frac{\partial T(\mathbf{r})}{\partial z}\right)^2 = \frac{1}{V(\mathbf{r})^2}, \quad (\text{F.2})$$

where  $\mathbf{r} = (r_x, r_y, r_z)$  is the position vector,  $V$  the velocity, and  $T(\mathbf{r})$  the travel-time for a ray generated by the source placed on the surface to arrive at point  $\mathbf{r}$ . The eikonal equation is discretized by finite differences and solved by the Fast Marching Method [20].

Iterative linearized approaches for solving the inverse problem for estimating  $V(\mathbf{r})$  resulting from (F.1) face problems related to non-linearity and ill-posedness. These methodologies usually produce inaccurate parameters due to convergence to local minima. Besides, iterative linearized methods are not able to properly quantify the uncertainty associated with the inversion. To address this issue, we have chosen BET that aggregates prior information to the inversion, a fact that also tends to regularize the problem, and informs the uncertainties associated with the estimated parameters. A Bayesian approach sets the parameters to be estimated as random variables to be characterized through their probability conditional distribution (PDF) with respect to observed data. The solution of the inverse problem is thus the posterior distribution  $p(\mathbf{m}|\mathbf{d}^{obs})$  phrased through Bayes' theorem as,

$$p(\mathbf{m}|\mathbf{d}^{obs}) = \frac{p(\mathbf{d}^{obs}|\mathbf{m})p(\mathbf{m})}{p(\mathbf{d}^{obs})}, \quad (\text{F.3})$$

where  $p(\mathbf{d}^{obs}|\mathbf{m})$  is the likelihood function,  $p(\mathbf{m})$  is the parameters prior, and  $p(\mathbf{d}^{obs})$  is the evidence.

The likelihood represents how well the estimated travel times fit the measurements. Assuming that the additive noise in Equation (F.1) is a Gaussian vector of independent and identically distributed components with zero mean and, also, that the variance  $\sigma_d^2$  is known, the likelihood is cast as,

$$p(\mathbf{d}^{obs}|\mathbf{m}) = \frac{1}{\sqrt{2\pi}\sigma_d} \exp\left(-\frac{1}{2}\left\|\frac{F(\mathbf{m}) - \mathbf{d}^{obs}}{\sigma_d}\right\|^2\right), \quad (\text{F.4})$$

where  $\|\cdot\|$  is the standard  $l_2$ -norm.

Due to analytical intractability of the posterior (F.3), resulting from the high dimensionality embedded in the likelihood, the non-availability of closed solutions for the eikonal equation, and the non-conjugated priors, we use an MCMC algorithm to produce samples from this posterior, yielding an ensemble that characterizes the uncertainty in the velocity field. A key point for the inversion is the choice of the parametrization of the discrete spatially heterogeneous velocity field. To cope with that, we propose a simultaneous identification of the velocity field and its dimensionality  $n$ . More specifically, following [12], we use a Voronoi's tessellation decomposing the domain into non-overlapping cells centered at the grid points with a constant value of the velocity in the cell interior. Such an assumption works as a prior that embodies the expected velocity field non-stationary character, allowing localized abrupt changes and continuous variation by choosing the number, position, and form of the cells. We explore the resulting posterior distribution by employing the RJMCMC method [12, 16, 53, 54], a generalization of classical MCMC samplers designed to handle, at each step of the algorithm, the seek for the appropriate dimension of the parameters space. The RJMCMC relies on the choice of convenient proposals and conditional priors, which is detailed in [12], and avoids the computation, as in traditional MCMC algorithms, of the denominator of Eq. (F.3), the evidence term. Besides the use of a burn-in phase for picking samples that are representative of the posterior distribution, to make such ensemble statistically meaningful to the sequence of workflow stages, we introduce an ad-hoc convergence checking procedure. We verify convergence in batches of increasing size if mean and variance of the estimated velocities achieve stable values. When that happens, we stop the algorithm, and this final batch constitutes the ensemble of uncertain velocity fields deployed to the next workflow stage.

## F.2 Assessing uncertainty in images

To create a global representation of the solution variability within the ensemble of plausible velocity fields and images produced by the previous stages of the workflow, we follow [73]. They proposed a confidence index, that normalizes the uncertainty map explored by LI *et al.* [71], that assigns at each spatial point an evaluation of the degree of uncertainty expressed by the standard deviation  $\sigma(\mathbf{r})$ , where low values represent regions where there are high variabilities, and high values the regions where there are small

variabilities. The confidence index is expressed by,

$$c(\mathbf{r}) = \frac{\sigma_{max} - \sigma(\mathbf{r})}{\sigma_{max} - \sigma_{min}}, \quad (\text{F.5})$$

where  $c(\mathbf{r})$  is the confidence in position  $\mathbf{r}$ , and  $\sigma_{min}$ ,  $\sigma_{max}$  and are the minimum, maximum and standard deviations in the  $\mathbf{r}$  position, respectively.

# Appendix G

## Profiling Reports

We show in this appendix the profiling reports for the seismic modeling (Appendix G.1) and RTM (Appendix G.2) on SX-Aurora TSUBASA and NVIDIA GPU platforms. proginf and ftrace are profiling tools provided by NEC corporation, and nvprof is a profiling tool from NVIDIA<sup>1</sup>. We use the profilers proginf, ftrace, and nvprof tools to get information about the computational performance of seismic applications. Discussions about the measurements can be seen in the numerical experiments chapter. Here, the reports are for consulting, which intends to support the discussion.

### G.1 Profiling analysis on SX-Aurora TSUBASA

EXCLUSIVE TIME[sec] ( % )	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU	PORT CONF	VLD HIT	LLC E.%	PROC.NAME
3.931 ( 99.1)	0.061	97316.1	45579.3	99.29	190.7	3.936	0.008	0.000	0.000	99.99	isotropicAcousticModeling\$2	
0.494 ( 12.4)	0.062	96980.7	45422.5	99.29	190.7	0.496	0.000	0.000	0.000	99.99	-thread0	
0.490 ( 12.4)	0.061	97617.9	45720.8	99.29	190.7	0.491	0.001	0.000	0.000	99.99	-thread1	
0.492 ( 12.4)	0.061	97342.7	45591.9	99.29	190.7	0.492	0.001	0.000	0.000	99.99	-thread2	
0.491 ( 12.4)	0.061	97437.2	45636.1	99.29	190.7	0.492	0.001	0.000	0.000	99.99	-thread3	
0.491 ( 12.4)	0.061	97543.9	45686.0	99.29	190.7	0.491	0.001	0.000	0.000	99.99	-thread4	
0.491 ( 12.4)	0.061	97515.5	45672.7	99.29	190.7	0.492	0.001	0.000	0.000	99.99	-thread5	
0.492 ( 12.4)	0.061	97363.5	45601.4	99.29	190.7	0.492	0.001	0.000	0.000	99.99	-thread6	
0.490 ( 12.4)	0.061	96727.8	45303.6	99.29	190.7	0.490	0.001	0.000	0.000	99.99	-thread7	
0.026 ( 0.7)	26.107	1533.9	0.1	58.91	245.7	0.004	0.007	0.001	100.00	isotropicAcousticModeling		
0.007 ( 0.2)	6.971	1150.1	0.0	0.00	0.0	0.000	0.000	0.000	0.00	main		
0.001 ( 0.0)	1.285	734.3	0.0	43.74	61.6	0.000	0.000	0.000	0.00	97.58	extend_matrix	
0.000 ( 0.0)	0.069	264.9	0.0	4.01	243.3	0.000	0.000	0.000	100.00	reading_file		
0.000 ( 0.0)	0.053	173.9	0.0	0.71	33.0	0.000	0.000	0.000	0.00	18.18	recording_file	
0.000 ( 0.0)	0.005	34566.8	0.0	98.14	240.0	0.000	0.000	0.000	0.00	56.31	extend_matrix\$1	
0.000 ( 0.0)	0.004	44602.2	0.0	98.15	240.0	0.000	0.000	0.000	100.00	-thread0		
0.000 ( 0.0)	0.005	33591.2	0.0	98.15	240.0	0.000	0.000	0.000	0.00	9.88	-thread1	
0.000 ( 0.0)	0.006	30974.5	0.0	98.14	240.0	0.000	0.000	0.000	0.00	40.67	-thread2	
0.000 ( 0.0)	0.005	33380.2	0.0	98.14	240.0	0.000	0.000	0.000	0.00	42.55	-thread3	
0.000 ( 0.0)	0.005	32756.7	0.0	98.13	240.0	0.000	0.000	0.000	0.00	41.20	-thread4	
0.000 ( 0.0)	0.005	39089.0	0.0	98.13	240.0	0.000	0.000	0.000	0.00	99.64	-thread5	
0.000 ( 0.0)	0.005	32913.3	0.0	98.13	240.0	0.000	0.000	0.000	0.00	39.96	-thread6	
0.000 ( 0.0)	0.005	32777.6	0.0	98.12	240.0	0.000	0.000	0.000	0.00	76.56	-thread7	
0.000 ( 0.0)	0.005	93516.0	46286.4	98.99	193.3	0.000	0.000	0.000	0.00	79.23	isotropicAcousticModeling\$1	
0.000 ( 0.0)	0.005	98925.9	48967.3	99.00	193.3	0.000	0.000	0.000	0.00	83.28	-thread0	
0.000 ( 0.0)	0.005	91479.9	45280.9	99.00	193.3	0.000	0.000	0.000	0.00	88.25	-thread1	
0.000 ( 0.0)	0.005	88471.8	43791.3	98.99	193.3	0.000	0.000	0.000	0.00	63.37	-thread2	
0.000 ( 0.0)	0.005	91241.0	45161.3	98.99	193.3	0.000	0.000	0.000	0.00	80.39	-thread3	
0.000 ( 0.0)	0.005	91906.9	45490.2	98.99	193.3	0.000	0.000	0.000	0.00	77.99	-thread4	
0.000 ( 0.0)	0.005	94775.4	46909.3	98.99	193.3	0.000	0.000	0.000	0.00	75.39	-thread5	
0.000 ( 0.0)	0.005	96576.1	47799.9	98.99	193.3	0.000	0.000	0.000	0.00	77.21	-thread6	
0.000 ( 0.0)	0.004	95976.5	47496.7	98.98	193.3	0.000	0.000	0.000	0.00	89.37	-thread7	
3.966 (100.0)	0.062	96479.8	45181.7	99.29	190.7	3.940	0.015	0.001	99.99	total		

Figure G.1: Performance information for the seismic modeling using the ftrace tool on SX-Aurora TSUBASA vector processor.

<sup>1</sup><https://docs.nvidia.com/cuda/profiler-users-guide/index.html>

EXCLUSIVE TIME[sec]( %)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU PORT	VLD CONF	LLC HIT	PROC.NAME
18.337 ( 87.3)	0.076	81136.8	44772.0	99.33	219.0	18.206	0.115	0.000	93.20	iso_acoustic_wave_equation\$1	
2.321 ( 11.0)	0.077	80418.1	44375.7	99.33	219.0	2.307	0.012	0.000	92.04	-thread0	
2.278 ( 10.8)	0.076	81921.2	45205.1	99.33	219.0	2.266	0.012	0.000	93.20	-thread1	
2.298 ( 10.9)	0.077	81208.9	44811.9	99.33	219.0	2.284	0.012	0.000	93.53	-thread2	
2.280 ( 10.9)	0.076	81850.7	45166.0	99.33	219.0	2.265	0.013	0.000	93.53	-thread3	
2.287 ( 10.9)	0.076	81608.5	45032.3	99.33	219.0	2.271	0.013	0.000	93.42	-thread4	
2.300 ( 10.9)	0.077	81158.7	44784.1	99.33	219.0	2.283	0.014	0.000	93.40	-thread5	
2.303 ( 11.0)	0.077	81031.0	44713.5	99.33	219.0	2.289	0.012	0.000	93.92	-thread6	
2.269 ( 10.8)	0.076	79903.3	44091.1	99.33	219.0	2.241	0.026	0.000	92.57	-thread7	
1.589 ( 7.6)	1588.930	397.6	3.3	0.00	117.4	0.000	0.000	0.000	100.00	random_damping_layers_carrilho	
0.881 ( 4.2)	0.011	56887.8	18753.3	98.90	187.0	0.812	0.064	0.000	39.22	cross_correlation\$1	
0.109 ( 0.5)	0.011	57721.1	19028.7	98.90	187.0	0.099	0.008	0.000	37.23	-thread0	
0.111 ( 0.5)	0.011	56421.7	18600.1	98.90	187.0	0.102	0.008	0.000	38.61	-thread1	
0.110 ( 0.5)	0.011	56709.6	18694.8	98.90	187.0	0.102	0.008	0.000	40.79	-thread2	
0.110 ( 0.5)	0.011	57190.9	18853.3	98.90	187.0	0.101	0.008	0.000	42.50	-thread3	
0.111 ( 0.5)	0.011	56513.6	18629.8	98.90	187.0	0.101	0.009	0.000	39.41	-thread4	
0.110 ( 0.5)	0.011	57094.1	18821.0	98.89	187.0	0.101	0.008	0.000	40.40	-thread5	
0.109 ( 0.5)	0.011	57217.2	18861.3	98.89	187.0	0.102	0.007	0.000	39.18	-thread6	
0.111 ( 0.5)	0.011	56264.1	18546.9	98.89	187.0	0.103	0.008	0.000	35.65	-thread7	
0.114 ( 0.5)	0.004	561.6	0.0	0.00	0.0	0.001	0.058	0.000	0.00	iso_acoustic_wave_equation	
0.036 ( 0.2)	0.004	2370.7	627.7	79.43	187.0	0.005	0.019	0.000	41.05	cross_correlation	
0.033 ( 0.2)	32.864	2197.2	900.9	81.67	245.7	0.018	0.009	0.001	0.62	adjointModeling	
0.009 ( 0.0)	9.212	1152.5	0.1	0.04	187.0	0.000	0.000	0.000	50.00	main	
0.005 ( 0.0)	4.666	1080.8	1.0	0.11	221.7	0.000	0.001	0.000	15.19	isotropicAcousticModeling	
0.000 ( 0.0)	0.039	86536.0	54592.5	99.14	184.3	0.000	0.000	0.000	100.00	laplacian_filter2D\$1	
0.000 ( 0.0)	0.039	86103.3	54320.0	99.14	184.3	0.000	0.000	0.000	100.00	-thread0	
0.000 ( 0.0)	0.038	88114.7	55588.8	99.14	184.3	0.000	0.000	0.000	100.00	-thread1	
0.000 ( 0.0)	0.040	85232.3	53770.3	99.14	184.3	0.000	0.000	0.000	100.00	-thread2	
0.000 ( 0.0)	0.038	88129.8	55598.1	99.14	184.3	0.000	0.000	0.000	100.00	-thread3	
0.000 ( 0.0)	0.039	86361.2	54482.2	99.14	184.3	0.000	0.000	0.000	99.98	-thread4	
0.000 ( 0.0)	0.040	85133.4	53707.5	99.14	184.3	0.000	0.000	0.000	100.00	-thread5	
0.000 ( 0.0)	0.039	87009.7	54891.1	99.14	184.3	0.000	0.000	0.000	100.00	-thread6	
0.000 ( 0.0)	0.038	86299.3	54442.5	99.13	184.3	0.000	0.000	0.000	100.00	-thread7	
0.000 ( 0.0)	0.007	62400.9	15396.3	98.69	187.0	0.000	0.000	0.000	49.67	main\$1	
0.000 ( 0.0)	0.007	63508.6	15670.5	98.70	187.0	0.000	0.000	0.000	49.82	-thread0	
0.000 ( 0.0)	0.007	62613.4	15449.4	98.70	187.0	0.000	0.000	0.000	49.72	-thread1	
0.000 ( 0.0)	0.007	62480.9	15416.4	98.70	187.0	0.000	0.000	0.000	49.11	-thread2	
0.000 ( 0.0)	0.007	62495.2	15419.7	98.69	187.0	0.000	0.000	0.000	49.34	-thread3	
0.000 ( 0.0)	0.007	59968.8	14796.1	98.69	187.0	0.000	0.000	0.000	49.74	-thread4	
0.000 ( 0.0)	0.007	62357.6	15385.2	98.69	187.0	0.000	0.000	0.000	49.97	-thread5	
0.000 ( 0.0)	0.007	62975.5	15537.4	98.69	187.0	0.000	0.000	0.000	49.69	-thread6	
0.000 ( 0.0)	0.007	62935.9	15527.4	98.69	187.0	0.000	0.000	0.000	49.95	-thread7	
0.000 ( 0.0)	0.006	86286.1	42764.6	99.12	221.7	0.000	0.000	0.000	0.03	adjointModeling\$1	
0.000 ( 0.0)	0.007	84925.7	42092.6	99.13	221.7	0.000	0.000	0.000	0.05	-thread0	
0.000 ( 0.0)	0.007	86617.0	42930.3	99.13	221.7	0.000	0.000	0.000	0.01	-thread1	
0.000 ( 0.0)	0.007	83890.1	41578.3	99.13	221.7	0.000	0.000	0.000	0.05	-thread2	
0.000 ( 0.0)	0.006	88940.6	44080.8	99.12	221.7	0.000	0.000	0.000	0.02	-thread3	
0.000 ( 0.0)	0.007	86347.9	42795.3	99.12	221.7	0.000	0.000	0.000	0.01	-thread4	
0.000 ( 0.0)	0.007	86583.6	42911.6	99.12	221.7	0.000	0.000	0.000	0.01	-thread5	
0.000 ( 0.0)	0.006	88469.6	43845.8	99.12	221.7	0.000	0.000	0.000	0.03	-thread6	
0.000 ( 0.0)	0.006	84577.0	41912.8	99.11	221.7	0.000	0.000	0.000	0.04	-thread7	
0.000 ( 0.0)	0.006	100108.8	49615.4	99.12	221.7	0.000	0.000	0.000	57.12	isotropicAcousticModeling\$1	
0.000 ( 0.0)	0.006	97339.7	48245.4	99.13	221.7	0.000	0.000	0.000	55.47	-thread0	
0.000 ( 0.0)	0.006	99967.8	49547.4	99.13	221.7	0.000	0.000	0.000	54.35	-thread1	
0.000 ( 0.0)	0.006	97497.0	48322.2	99.13	221.7	0.000	0.000	0.000	38.77	-thread2	
0.000 ( 0.0)	0.006	103302.6	51199.0	99.12	221.7	0.000	0.000	0.000	65.14	-thread3	
0.000 ( 0.0)	0.006	102046.5	50575.8	99.12	221.7	0.000	0.000	0.000	66.54	-thread4	
0.000 ( 0.0)	0.006	99163.6	49146.4	99.12	221.7	0.000	0.000	0.000	66.95	-thread5	
0.000 ( 0.0)	0.006	102311.1	50705.7	99.12	221.7	0.000	0.000	0.000	57.74	-thread6	
0.000 ( 0.0)	0.005	99519.1	49317.6	99.11	221.7	0.000	0.000	0.000	51.36	-thread7	
0.000 ( 0.0)	0.005	43727.7	0.0	97.93	187.0	0.000	0.000	0.000	60.19	random_damping_layers_carr	
0.000 ( 0.0)	0.004	50933.0	0.0	97.94	187.0	0.000	0.000	0.000	100.00	-thread0	
0.000 ( 0.0)	0.004	49235.2	0.0	97.94	187.0	0.000	0.000	0.000	82.98	-thread1	
0.000 ( 0.0)	0.005	44042.0	0.0	97.93	187.0	0.000	0.000	0.000	79.20	-thread2	
0.000 ( 0.0)	0.005	43925.5	0.0	97.93	187.0	0.000	0.000	0.000	8.18	-thread3	
0.000 ( 0.0)	0.006	38256.0	0.0	97.93	187.0	0.000	0.000	0.000	100.00	-thread4	
0.000 ( 0.0)	0.005	44079.3	0.0	97.92	187.0	0.000	0.000	0.000	4.48	-thread5	
0.000 ( 0.0)	0.005	43579.6	0.0	97.92	187.0	0.000	0.000	0.000	7.18	-thread6	
0.000 ( 0.0)	0.005	38818.3	0.0	97.92	187.0	0.000	0.000	0.000	99.50	-thread7	
0.000 ( 0.0)	0.010	3171.8	1911.1	86.37	250.1	0.000	0.000	0.000	0.00	ricker_wavelet	
0.000 ( 0.0)	0.004	10270.3	5259.0	95.10	184.3	0.000	0.000	0.000	100.00	laplacian_filter2D	
0.000 ( 0.0)	0.004	28770.2	0.0	93.09	40.0	0.000	0.000	0.000	0.00	cleaningImage	
21.005 (100.0)	0.058	73261.6	39876.4	99.26	217.9	19.043	0.268	0.001	91.14	total	

Figure G.2: Performance information for the RTM using the ftrace tool on SX-Aurora TSUBASA vector processor.

## G.2 Profiling analysis on NVIDIA GPU

```

==18925== Profiling result:
Type      Type      Time      Calls      Avg      Min      Max      Name
GPU activities:
09.75%    1.69289s    8001    211.57us    203.92us    269.15us    isotropicAcousticModeling 84 gpu
28.99%    703.62ms    8001    87.94us    86.85us    98.15us    isotropicAcousticModeling 108 gpu
1.06%    25.68ms    8001    3.2100us    2.7520us    17.69us    isotropicAcousticModeling 96 gpu
0.14%    3.4872ms    1    3.4872ms    3.4872ms    3.4872ms    [CUDA memcpy DtoH]
0.65%    1.1842ms    500    2.3680us    2.2400us    12.96us    isotropicAcousticModeling 103 gpu
0.01%    308.44us    2    154.22us    2.1760us    306.27us    [CUDA memcpy HtoD]
API calls:
78.91%    2.39294s    8003    298.89us    81.44us    3.4882ms    cuStreamSynchronize
14.83%    449.52ms    1    449.52ms    449.52ms    449.52ms    cuDevicePrimaryCtxRetain
4.55%    137.80ms    1    137.80ms    137.80ms    137.80ms    cuMemHostAlloc
1.62%    49.226ms    24503    2.0080us    1.7050us    215.24us    cuLaunchKernel
0.06%    1.7610ms    1    1.7610ms    1.7610ms    1.7610ms    cuMemAllocHost
0.02%    737.62us    7    105.37us    5.6740us    381.94us    cuMemAlloc
0.01%    333.70us    1    333.70us    333.70us    333.70us    cuModuleLoadDataEx
0.00%    49.652us    1    49.652us    49.652us    49.652us    cuStreamCreate
0.00%    18.681us    2    9.3400us    3.3680us    15.313us    cuMemcpyHtoDAsync
0.00%    10.789us    1    10.789us    10.789us    10.789us    cuMemcpyDtoHAsync
0.00%    9.1510us    9    1.0160us    258ns    4.2700us    cuPointerGetAttributes
0.00%    8.9820us    1    8.9820us    8.9820us    8.9820us    cuDeviceGetPCIBusId
0.00%    6.3110us    3    2.1030us    1.2290us    2.7290us    cuCtxSetCurrent
0.00%    3.0940us    5    618ns    219ns    1.3750us    cuDeviceGetAttribute
0.00%    2.4340us    4    608ns    207ns    1.6260us    cuModuleGetFunction
0.00%    2.3850us    2    1.1920us    494ns    1.8910us    cuDeviceTotalMem
0.00%    1.8030us    3    601ns    251ns    1.2270us    cuDeviceGetCount
0.00%    952ns    2    476ns    214ns    738ns    cuDeviceGet
0.00%    488ns    1    488ns    488ns    488ns    cuCtxGetCurrent
0.00%    429ns    1    429ns    429ns    429ns    cuDeviceComputeCapability
0.00%    293ns    1    293ns    293ns    293ns    cuDriverGetVersion
OpenACC (excl):
95.37%    2.39375s    8001    299.10us    82.353us    447.40us    acc_wait@forward_modeling.c:108
0.95%    23.672ms    8001    2.9580us    2.5950us    216.71us    acc_enqueue_launch@forward_modeling.c:84 (isotropicAcousticModeling 84 gpu)
0.90%    22.255ms    8001    2.7820us    2.5140us    210.22us    acc_enqueue_launch@forward_modeling.c:96 (isotropicAcousticModeling 96 gpu)
0.86%    21.315ms    8001    2.6640us    2.4000us    255.68us    acc_enqueue_launch@forward_modeling.c:108 (isotropicAcousticModeling 108 gpu)
0.68%    16.840ms    8001    2.1040us    1.8940us    211.87us    acc_compute_construct@forward_modeling.c:81
0.14%    3.4889ms    1    3.4889ms    3.4889ms    3.4889ms    acc_wait@forward_modeling.c:116
0.06%    1.3754ms    500    2.7500us    2.4590us    27.263us    acc_enqueue_launch@forward_modeling.c:103 (isotropicAcousticModeling 103 gpu)
0.02%    439.01us    3    146.34us    95.336us    188.08us    acc_enter_data@forward_modeling.c:77
0.02%    389.79us    1    389.79us    389.79us    389.79us    acc_device_init
0.01%    297.25us    1    297.25us    297.25us    297.25us    acc_wait@forward_modeling.c:77
0.00%    36.911us    3    12.303us    2.6220us    25.961us    acc_exit_data@forward_modeling.c:77
0.00%    24.509us    2    12.254us    4.0990us    20.410us    acc_enqueue_upload@forward_modeling.c:77
0.00%    11.976us    1    11.976us    11.976us    11.976us    acc_enqueue_download@forward_modeling.c:116
0.00%    0ns    6    0ns    0ns    0ns    acc_alloc@forward_modeling.c:77
0.00%    0ns    6    0ns    0ns    0ns    acc_create@forward_modeling.c:77
0.00%    0ns    6    0ns    0ns    0ns    acc_delete@forward_modeling.c:116

```

Figure G.3: Seismic modeling profiling data of CUDA-related activities on both CPU and GPU using the nvprof tool.

```

==22351== Profiling result:
Type      Type      Time      Calls      Avg      Min      Max      Name
GPU activities:
42.84%    4.99089s    10001    498.96us    498.29us    508.43us    adjointModeling 101 gpu
21.90%    2.55100s    10001    255.07us    248.25us    329.15us    isotropicAcousticModeling 83 gpu
18.69%    2.17776s    10001    217.75us    217.20us    236.12us    adjointModeling 155 gpu
9.29%    1.08205s    10001    108.19us    107.21us    119.88us    isotropicAcousticModeling 103 gpu
6.85%    798.18ms    10001    79.810us    79.399us    101.32us    adjointModeling 147 gpu
0.35%    41.070ms    10001    4.1060us    3.4880us    13.793us    adjointModeling 175 gpu
0.06%    6.8363ms    6    1.1394ms    2.7200us    5.2366ms    [CUDA memcpy HtoD]
0.01%    1.2986ms    500    2.5970us    2.4640us    13.18us    isotropicAcousticModeling 98 gpu
0.01%    929.85us    3    309.95us    252.12us    339.23us    [CUDA memcpy DtoH]
API calls:
97.20%    11.5649s    20006    578.07us    138.56us    6.0976ms    cuStreamSynchronize
1.01%    120.51ms    60506    1.9910us    1.6440us    200.77us    cuLaunchKernel
0.90%    107.05ms    1    107.05ms    107.05ms    107.05ms    cuMemHostAlloc
0.87%    103.94ms    1    103.94ms    103.94ms    103.94ms    cuDevicePrimaryCtxRetain
0.01%    1.0609ms    11    96.446us    5.5360us    410.49us    cuMemAlloc
0.00%    380.14us    1    380.14us    380.14us    380.14us    cuMemAllocHost
0.00%    80.052us    1    80.052us    80.052us    80.052us    cuModuleLoadDataEx
0.00%    45.569us    6    7.5940us    1.0970us    10.152us    cuMemcpyHtoDAsync
0.00%    23.265us    3    7.7550us    2.4500us    11.239us    cuMemcpyDtoHAsync
0.00%    19.505us    19    1.0260us    213ns    3.7090us    cuPointerGetAttributes
0.00%    13.488us    1    13.488us    13.488us    13.488us    cuStreamCreate
0.00%    10.526us    1    10.526us    10.526us    10.526us    cuDeviceGetPCIBusId
0.00%    3.4450us    7    492ns    132ns    989ns    cuModuleGetFunction
0.00%    2.2710us    3    757ns    232ns    1.7680us    cuDeviceGetCount
0.00%    1.8620us    5    372ns    189ns    681ns    cuDeviceGetAttribute
0.00%    1.2110us    3    403ns    215ns    583ns    cuCtxSetCurrent
0.00%    1.0510us    2    525ns    179ns    872ns    cuDeviceGet
0.00%    822ns    2    411ns    85ns    737ns    cuDeviceTotalMem
0.00%    494ns    1    494ns    494ns    494ns    cuCtxGetCurrent
0.00%    452ns    1    452ns    452ns    452ns    cuDeviceComputeCapability
0.00%    262ns    1    262ns    262ns    262ns    cuDriverGetVersion
OpenACC (excl):
67.38%    7.94699s    10001    794.62us    546.03us    1.0116ms    acc_wait@backward_modeling.c:175
30.74%    3.62484s    10001    362.45us    139.50us    492.82us    acc_wait@forward_modeling.c:103
0.27%    31.284ms    10001    3.1280us    2.6350us    232.53us    acc_enqueue_launch@backward_modeling.c:101 (adjointModeling 101 gpu)
0.25%    29.423ms    10001    2.9410us    2.5790us    267.89us    acc_enqueue_launch@forward_modeling.c:83 (isotropicAcousticModeling 83 gpu)
0.23%    27.327ms    10001    2.7320us    2.4500us    216.60us    acc_enqueue_launch@backward_modeling.c:147 (adjointModeling 147 gpu)
0.23%    27.301ms    10001    2.7290us    2.3920us    218.36us    acc_enqueue_launch@forward_modeling.c:103 (isotropicAcousticModeling 103 gpu)
0.23%    26.983ms    10001    2.6970us    2.3830us    238.24us    acc_enqueue_launch@backward_modeling.c:155 (adjointModeling 155 gpu)
0.22%    26.126ms    10001    2.6120us    2.3900us    218.85us    acc_enqueue_launch@backward_modeling.c:175 (adjointModeling 175 gpu)
0.22%    25.965ms    10001    2.5960us    2.3770us    19.509us    acc_compute_construct@backward_modeling.c:98
0.15%    17.156ms    10001    1.7150us    1.5250us    217.22us    acc_compute_construct@forward_modeling.c:80
0.05%    6.0602ms    1    6.0602ms    6.0602ms    6.0602ms    acc_wait@backward_modeling.c:92

```

Figure G.4: RTM profiling data of CUDA-related activities on both CPU and GPU using the nvprof tool.