GPU Performance analysis for viscoacoustic wave equations using fast stencil computation from the symbolic specification

Lauê Jesus
Supercomputing Center for Industrial Innovation, SENAI/CIMATEC
Peterson Nogueira
Supercomputing Center for Industrial Innovation, SENAI/CIMATEC
João Speglich
Supercomputing Center for Industrial Innovation, SENAI/CIMATEC
Murilo Boratto (murilo.boratto@fieb.org.br)
Supercomputing Center for Industrial Innovation, SENAI/CIMATEC

Research Article

Keywords: Viscoacoustic wave equations, Finite-Difference, Devito, GPU

Posted Date: September 12th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2039437/v1

GPU Performance analysis for viscoacoustic wave equations using fast stencil computation from the symbolic specification

Lauê Jesus[1], Peterson Nogueira[1], João Speglich[1] and Murilo Boratto[1*]

[1]Supercomputing Center for Industrial Innovation, SENAI/CIMATEC, Av. Orlando Gomes, 1845 - Piatã, Salvador, 41650-010, Bahia, Brazil.

*Corresponding author(s). E-mail(s): murilo.boratto@fieb.org.br; Contributing authors: laue.jesus@fieb.org.br; peterson.santos@fieb.org.br; joao.speglichr@fieb.org.br;

Abstract

Seismic forward modeling is a computationally and data-intensive stage in the seismic processing workflow. By profiling the kernels of seismic forward modeling algorithms, was observed that they need to access a wide variety of memory locations, in addition to the computational cost of performing floating-point operations for the numerical solution of wave equations. In this context, was used the Roofline model to analyze six representative computing kernels in seismic modeling on GPU environment to indicate bottlenecks in the performance and suggest improvements of these wave equation propagators. Based on this, was implemented six viscoacoustic equations using the Devito tool. Experimental data have shown that optimizations in increasing data reuse and decreasing off-chip memory traffic can significantly improve performance.

Keywords: Viscoacoustic wave equations, Finite-Difference, Devito, GPU

1 Introduction

Stencil computation is the essence of several applications in the high-performance computing area, such as computational fluid dynamics, image processing, and geophysical applications that involve solving wave equations. According to [1] stencil computations are often used in applications to solve partial differential equations in multidimensional grids. A typically 3D grid is iteratively traversed from a starting cell in stencil computation. Each grid cell is updated based on a set of coefficients and the values of its neighboring cells [2]. Regular and predictable memory access patterns characterize stencil computation; however, it has a low FLOP/byte ratio, precisely tuning the data layout and optimizing the memory accesses according to the different hardware architectures [3]. In the context of a wave equation solver, these are finite-difference (FD) schema time-marching kernels that update a data set in a grid at iteration n to get that of the next iteration.

The most common seismic modeling uses acoustic wave equation. However, several other wave equations which best represent the subsurface can be used for seismic modeling. Viscoacoustic modeling provides an accurate image with the high resolution of exploratory targets due to the inclusion of characteristics such as amplitude attenuation and velocity dispersion in the propagation of seismic waves. In this work, the viscoacoustic equations based on the Maxwell, Kelvin-Voigt, and standard linear solid (SLS) rheological models are described, considering first and second-order formulations. These equations are in the time-space domain, and the finite-difference method is suitable to solve them.

Our implemented FD kernel viscoacoustic equations were introduced into Devito [4], [5] and [6], an open-source domain-specific language (DSL) tool for highly optimized FD kernel solutions. Devito is available as a Python language package and can be integrated with different packages, such as SymPy [7] and Numpy [8].

The methodology used includes using the Roofline model [9], [10], and [11], which proposes to identify performance bottlenecks and be a guide to performance optimization. According to [11], the Roofline is a performance-oriented model centered around computational capabilities, memory bandwidth, and data locality. The data locality is commonly expressed as the arithmetic intensity (AI), the ratio of floating-point operations performed to data moved (FLOP/Byte).

The present work describes the development and implementation of six viscoacoustic equations on the Devito tool. Through the Roofline model, the computational aspects of viscoacoustic seismic modeling will be evaluated. The primary purpose is to provide information about the bottlenecks and optimization strategies indicated from the computational analysis of the kernels of viscoacoustic equations. This paper proceeds as follows: Section 2 brings information on mathematical model to viscoacoustic wave equations. Section 3 details the implementations of our model over the parallel architectures. Section 4 presents our case studies and experimental results. Some conclusions and future work ideas close the paper.

2 Mathematical model to viscoacoustic wave equations

The viscoacoustic equations based on the rheological models originate from the stress-strain relation

$$\sigma = \frac{\partial \psi}{\partial t} * \epsilon = \psi * \frac{\partial \epsilon}{\partial t}, \tag{1}$$

where σ is the stress, ϵ is the deformation, ψ is the relaxation function. Moreover,

$$\frac{\partial \epsilon}{\partial t} = \nabla \cdot \mathbf{v}, \tag{2}$$

and

$$\frac{\partial \mathbf{v}}{\partial t} = \frac{1}{\rho}\nabla \sigma, \tag{3}$$

where $\mathbf{v}$ is the particle velocity, ρ is the density. Equations (1), (2), and (3) are the main relations to derive the viscoacoustic equations based on Maxwell, Kelvin-Voigt, and SLS rheological models [12].

2.1 Equations based on the Maxwell model

The relaxation function for the Maxwell model is defined as

$$\psi = M_U e^{(-t/\tau)} H(t), \tag{4}$$

where M_U is the elasticity constant of the unrelaxed spring, $H(t)$ is the Heaviside function and $\tau = \eta / M_U = \omega_0 Q$ is the relaxation time, being η the viscosity and Q the quality factor. Replacing (4) in (1), using (2) and (3), considering $M_U \approx k$ to the dominant frequency, and adding the source term S, derived the first-order equation

$$\begin{cases} \frac{\partial p}{\partial t} + \kappa \nabla \cdot \mathbf{v} + \frac{\omega_0}{Q} p = \int S(\mathbf{x}_s, t), \\ \frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho} \nabla P = 0, \end{cases} \tag{5}$$

where κ is the bulk modulus, $p = -\sigma$ is the pressure field, and $\omega_0 = 2\pi f_0$ is the angular frequency, being f_0 the dominant frequency.

Taking the time derivative in (5), and replacing the second term in the first term, derived the second-order equation

$$\frac{\partial^2 p}{\partial t^2} - \kappa \nabla \cdot \frac{1}{\rho} \nabla p + \frac{\omega_0}{Q} \frac{\partial p}{\partial t} = S(\mathbf{x}_s, t). \tag{6}$$

2.2 Equations based on the Kelvin-Voigt model

The relaxation function for the Kelvin-Voigt model is defined as

$$\psi = M_R H(t) + \eta \delta(t), \tag{7}$$

where M_R is the elasticity constant of the relaxed spring, η is the viscosity, $H(t)$ and $\delta(t)$ are the Heaviside and Dirac delta functions, respectively. Replacing (7) in (1), taking the time derivative, and using (2), is obtained

$$\frac{\partial \sigma}{\partial t} = M_R \nabla \cdot \mathbf{v} + \eta \nabla \cdot \frac{1}{\rho} \nabla \sigma. \tag{8}$$

Thus, considering $\sigma = -p$, the motion equation in (3), and adding the source term S, derived the first-order equation

$$\begin{cases} \frac{\partial p}{\partial t} + \kappa \nabla \cdot \mathbf{v} - \eta \nabla \cdot \frac{1}{\rho} \nabla p = \int S(\mathbf{x}_s, t), \\ \frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho} \nabla p = 0. \end{cases} \tag{9}$$

being $M_R \approx \kappa$, $\eta = \tau M_R$ with $\tau = (\omega_0 Q)^{-1}$, where κ, η, τ are the bulk modulus, viscosity, and relaxation time, respectively. Taking the time derivative in (9), and replacing the second term in the first term, derived the second-order equation

$$\frac{\partial^2 p}{\partial t^2} = \kappa \nabla \cdot \frac{1}{\rho} \nabla p + \eta \nabla \cdot \frac{1}{\rho} \nabla \frac{\partial p}{\partial t}. \tag{10}$$

2.2.1 Equations based on the SLS model

The relaxation function for the SLS model is defined as

$$\psi = \kappa \left[1 + \left(1 - \frac{\tau_\epsilon}{\tau_\sigma} \right) e^{-t/\tau_\sigma} \right] H(t). \tag{11}$$

Taking the time derivative in (1), and replacing (11) in (1), is obtained the relationship between the pressure field and particle velocity (3) [13], [14]:

$$\frac{\partial p}{\partial t} = -\frac{\partial \left[\kappa \left(1 + \tau e^{-\frac{t}{\tau_\sigma}} \right) H(t) \right]}{\partial t} * \nabla \cdot \mathbf{v} + S, \tag{12}$$

where $\kappa = \kappa(\mathbf{x})$ the bulk modulus, $p = -\sigma$ is the pressure field, $H(t)$ is the Heaviside function, and $S = S(\mathbf{x}_s, t)$ the source term at position $\mathbf{x}_s$. The $\tau = \tau_\epsilon / \tau_\sigma - 1$ represents the magnitude of Q. The τ_ϵ and τ_σ are stress and strain relaxation parameters given by:

$$\tau_\sigma = \frac{\sqrt{Q^2 + 1} - 1}{2\pi f_0 Q} \text{ and } \tau_\epsilon = \frac{\sqrt{Q^2 + 1} + 1}{2\pi f_0 Q}. \tag{13}$$

Equation (12) is expensive to solve by numerical modeling because of the associated convolution operation. So, it is necessary the introduction of a memory variable r_p [15], allowing us to derive the first-order equation

$$\begin{cases} \frac{\partial p}{\partial t} + \kappa(\tau+1)(\nabla \cdot \mathbf{v}) + r_p = \int S(\mathbf{x}_s, t), \\ \frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho}\nabla p = 0, \\ \frac{\partial r_p}{\partial t} + \frac{1}{\tau_\sigma}[r_p + \tau\kappa(\nabla \cdot \mathbf{v})] = 0. \end{cases} \tag{14}$$

For the second-order formulation, the derivative of time was taken in (12) and introducing r_p to remove the convolution term [16], resulting in

$$\begin{cases} \frac{1}{v^2}\frac{\partial^2 p}{\partial t^2} = (1+\tau)\rho\nabla \cdot \frac{1}{\rho}\nabla p - r_p + S(\mathbf{x}_s, t), \\ \frac{\partial r_p}{\partial t} = \frac{\tau}{\tau_\sigma}\rho\nabla \cdot \frac{1}{\rho}\nabla p - \frac{1}{\tau_\sigma} r_p. \end{cases} \tag{15}$$

3 Methodology

3.1 Hardware and software setup

All experiments performed in this work were run on a environment offering GPU node, where this node contains two 18-core Intel(R) Xeon(R) Gold 6240 CPU @2.60 GHz, 376 GB RAM, and 1 NVIDIA TESLA V100 SXM2 [17]. This architecture provides 80 streaming multiprocessors with 32 GB HBM2 memory. For the Roofline analysis, the *nvprof* tool [18] collects and visualizes the data obtained from the execution of kernel. The *nvprof* collects a timeline of application-related activities on the CPU and GPU, including kernel execution and memory transfers.

3.2 Experimental setup

As input parameters, the three layers seismic model Figure 1(a), with grid spacing in x, y, and z, and border size of equal to 8 and 50 meters (m), respectively. We performed the experiment considering two different grid point models. The first with 100^3 points and the second with 500^3 points. The choice of the model is for computational purposes only. In this work, we do not intend to evaluate the geophysical side. We generated a seismic shot Figure 1(b) with 4s of record time.

In this experiment, the source is located in the center position of the model surface. The receivers are spread along a plane (x, y). Such receivers capture energy from various points, which improves the illumination of the target explored. In this context, the seismogram of a shot with three-dimensional modeling provides details by taking planes along the x or y axes and analyzing them separately.

3.3 Roofline implementation

For the implementation of the Roofline model, the *nvprof* provides the data to feed the graph with information on AI and GFLOPs. First, it is

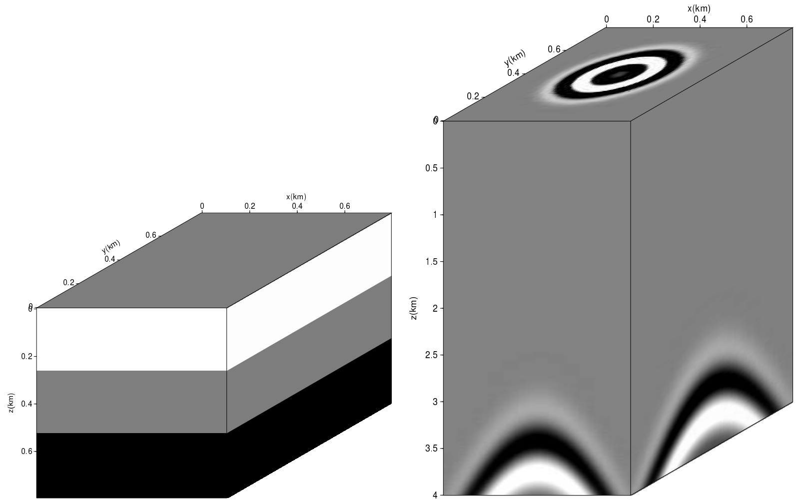


Figure 1 Seismic Forward Modeling: (a) 100×100×100 three layers seismic model. (b) 3D seismogram modeled with first-order SLS viscoacoustic equation, in the seismic central of model position (x = 400m and y = 400m), 1001 time steps.

necessary to perform the machine characterization to define the peak performance (GFLOP/s) and the bandwidth (GB/s), both attributes of the roofline graph. Based on that, was used the Empirical Roofline Toolkit (ERT) [19], whose function is to characterize GPU-accelerated systems. According to the methodology adopted in [11], to calculate the AI of the kernels, as well as the performance in GFLOP/s, it is necessary the use of the *nvprof* to obtain three basic information. The kernel execution time, the number of floating-point operations performed and the number of bytes read and write each memory level. From this:

$$\begin{cases} AI = \frac{\text{nvprof GFLOPs}}{\text{nvprof Bytes}}, \\ \\ GFLOP/s = \frac{\text{nvprof GFLOPs}}{\text{nvprof Runtime}}. \end{cases} \tag{16}$$

The Roofline plots that will be shown in the graphs are: The *p kernel*, *vx:vy:vz kernel* and *p sub-kernel*. The experiment has as main objective to measure the performance of these kernels. The *p kernel* represents the *p* pressure field update, present in the Kelvin-Voigt and Maxwell equations. The *p:r kernel* represents the pressure field *p* and the memory variable *r* updates (Algorithm 1), present in the SLS equation. The *vx:vy:vz kernel* represents the particle velocity components (vx, vy, and vz) update (Algorithm 2), present in the first-order viscoacoustic equations. The *p sub-kernel* represents optimizations that are inserted by the Devito compiler to eliminate redundancies in consecutive loop iterations and data reuse.

```
# The stress relaxation parameter
t_s = (sp.sqrt(1.+1./qp**2)-1./qp)/f0
# The strain relaxation parameter
t_ep = 1./(f0**2*t_s)
# The relaxation time
tt = (t_ep/t_s)-1.
# Bulk modulus
bm = rho * vp**2

pde_r = r - s * (1. / t_s) * r - s * (1. / t_s) * tt * rho
    * div(v.forward)
u_r = Eq(r.forward, damp * pde_r)

pde_p = p - s * bm * (tt + 1.) * div(v.forward) - s * vp**2
     * r.forward + s * vp**2 * q
u_p = Eq(p.forward, damp * pde_p)
```

Algorithm 1 Symbolic expression for the pressure field p and the memory variable r updates for SLS first-order viscoacoustic equation.

```
pde_v = v - s * b * grad(p)
u_v = Eq(v.forward, damp * pde_v)
```

Algorithm 2 Symbolic expression for the particle velocity v update for SLS first-order viscoacoustic equation.

3.4 Viscoacoustic wave equations implementation on Devito

3.4.1 Target application: seismic forward modelling workflow

Based on the seismic forward modeling application Figure 2, was had the seismic model definition and the acquisition geometry, which involves the source type and source peak frequency, as well as the source/receivers locations. Then the FD Kernel represents the function with the highest computational cost of the entire application. In the case of this application, for each viscoacoustic wave equation presented in Section 2 an FD Kernel has been implemented.

3.4.2 Case study: code generation of the first-order SLS viscoacoustic equation on Devito.

In the Algorithm 3 it is possible to visualize only a part of the automatically generated code, referring to the update of the particle velocity components of the first-order SLS viscoacoustic equation.

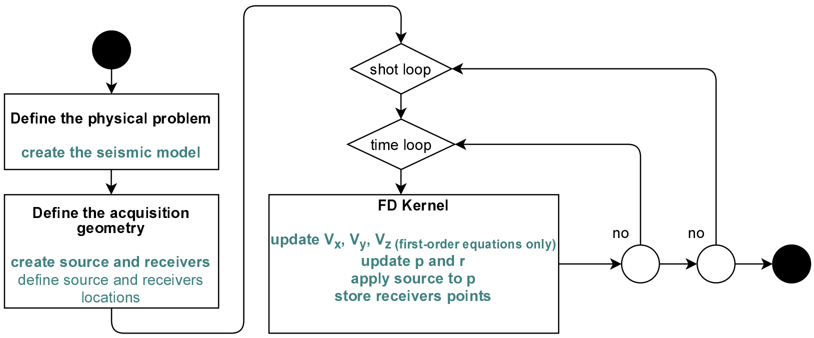


Figure 2 Forward modelling flowchart.

```
#pragma acc parallel loop collapse(3) present(b,p,vx,vy,vz)
for (int x = x_m; x <= x_M; x += 1)
{
 for (int y = y_m; y <= y_M; y += 1)
 {
  for (int z = z_m; z <= z_M; z += 1)
  {
   float r9 = -p[t0][x+16][y+16][z+16];
   vx[t1][x+16][y+16][z+16]=(-5.0e-1F*dt*(b[x+16][y+16][z
       +16]+ ...
   vy[t1][x+16][y+16][z+16]=(-5.0e-1F*dt*(b[x+16][y+16][z
       +16]+ ...
   vz[t1][x+16][y+16][z+16]=(-5.0e-1F*dt*(b[x+16][y+16][z
       +16]+ ...
  }
 }
}
```

Algorithm 3 The 3D grid point model vx:vy:vz to kernel.

3.4.3 Optimizations

For GPU flow, Devito presents two optimization levels (*noop*, *advanced*) for code generation. With *noop*, no performance optimizations are introduced. The *advanced* directive provides several flop-reducing and data locality optimization passes.

Cross-iteration redundancy elimination (CIRE) searches for redundancies across consecutive loop iterations. These are often induced by a mixture of nested, high-order, partial derivatives. The *advanced* mode has a code motion pass. In explicit PDE solvers, this is most commonly used to lift expensive time-invariant sub-expressions out of the inner loops. The *advanced* directive is used by default in Devito. However, the generated code still has optimization gaps and this work aimed to contribute to this aspect.

From the Algorithm 3 analysis and the AI and performance in GFLOP/s information obtained, it is assumed that the FD Kernel presents a computationally-intensive process with a large number of memory accesses and that demands a high memory bandwidth. In this context, the FD Kernel operates in a DRAM bandwidth limit region. Thus, strategies that optimize memory access are analyzed to take advantage of all memory bandwidth and eliminate redundant memory accesses.

There is a feature available in the standard OpenACC directives [20], which is the addition of the *tile* clause to the *acc loop* directive. With the *tile* clause, it is possible to optimize the loop by operating smaller blocks to explore locality and data reuse [21]. Using the *tile* clause in Devito is optional and is disabled by default.

According to [22] tiling can improve parallel stencil applications in at least 3 ways. First, tiling partitions loop data and computations into tiles, thereby enabling the GPU to handle amounts of input data that exceed the capacity of its internal memory. Second, tiling reorders loop nesting of the stencil, which can improve spatial and temporal locality within the tiles. Third, the partitioning combined with loop reordering potentially reduces the volume of communication between host memory and GPU, which reduces the memory bandwidth requirements for the application.

4 Experimental results

This section presents some results obtained with the usage of our proposed. Experimental results are shown and commented, with detailed explanation and useful insights.

4.1 First-order viscoacoustic equations

For each first-order viscoacoustic equation we performed application profiling. Figure 3 represents an approximate average of the execution time of the application functions using the first-order viscoacoustic equations.

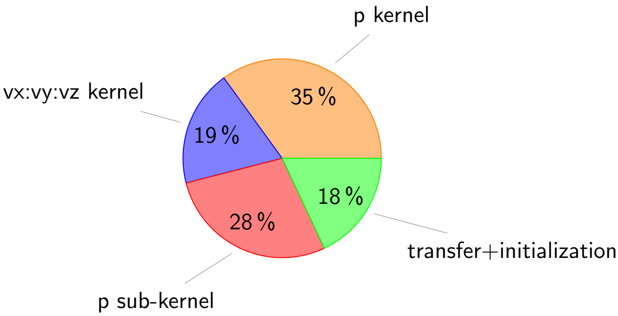
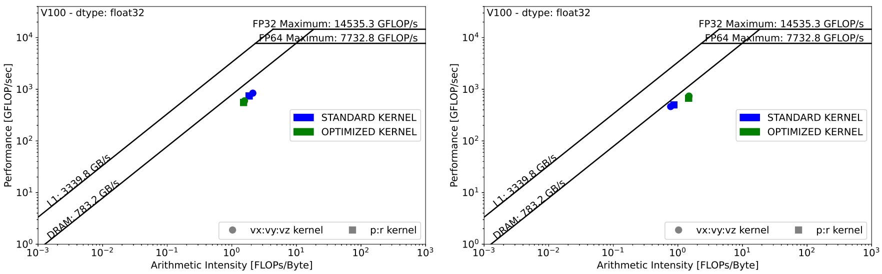


Figure 3 Application profiling using first-order viscoacoustic equation. High cost in terms of execution time.

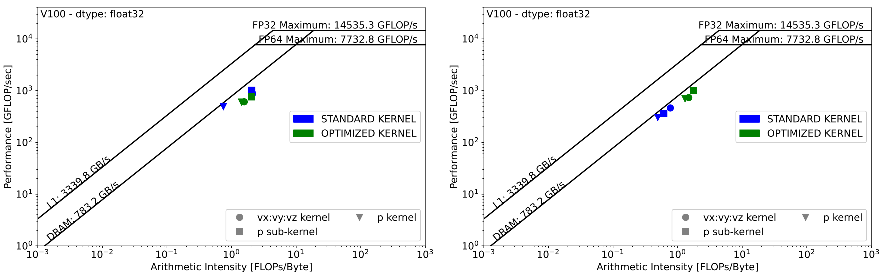
In the graphics of Figures 4, 5, and 6 was noticed similarity, the Roofline plots are located in a region where GFLOP/s performance is limited by memory bandwidth. From the tests performed, was noted in Figures 4 (b), 5 (b) and 6 (b) that the increase in the domain size results in the approximation between the Roofline plots and the memory bandwidth limit.

It is also important to note a better performance using the *tile* clause represented by `OPTIMIZED KERNEL`. Was understood that the optimization decrease the data movement between DRAM memory and device, which resulted in increased performance in GFLOP/s, Figures 4 (b), 5 (b), and 6 (b).



(a) (b)

Figure 4 Roofline first-order SLS viscoacoustic equation: (a) Roofline plots for a 100×100×100 model. (b) Roofline plots for a 500×500×500 model.



(a) (b)

Figure 5 Roofline first-order Kelvin-Voigt viscoacoustic equation: (a) Roofline plots for a 100×100×100 model. (b) Roofline plots for a 500×500×500 model.

4.2 Second-order viscoacoustic equation

For each second-order viscoacoustic equation we performed application profiling. Figure 7 represents an approximate average of the execution time of the application functions using the second-order viscoacoustic equations.

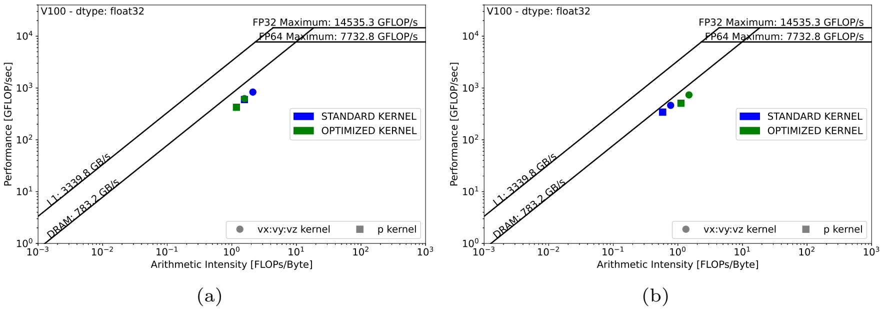


Figure 6 Roofline first-order Maxwell viscoacoustic equation: (a) Roofline plots for a 100×100×100 model. (b) Roofline plots for a 500×500×500 model.

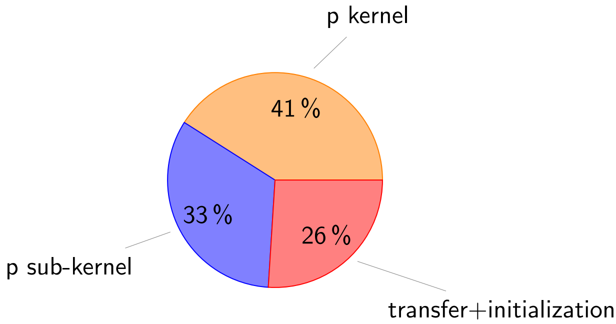


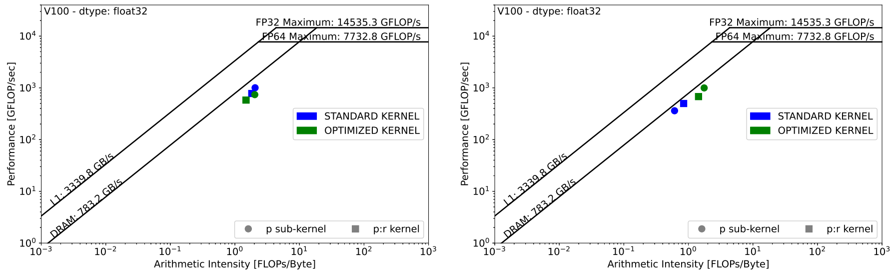
Figure 7 Application profiling using second-order viscoacoustic equation. High cost in terms of execution time.

In the graphics of Figures 8, 9, and 10 was noted that the Roofline plots are located in a region where GFLOP/s performance is limited by memory bandwidth. It is important to note that the performance increase with the use of the clause *tile* represented by the `OPTIMIZED KERNEL` was more significant for the second-order viscoacoustic equations compared to first-order viscoacoustic equations.

Figure 11 shows the seismic forward modeling execution time results, using each of the viscoacoustic equations. The input parameters are the same previously used. Was used the number of 10001 time steps so that the experiment.

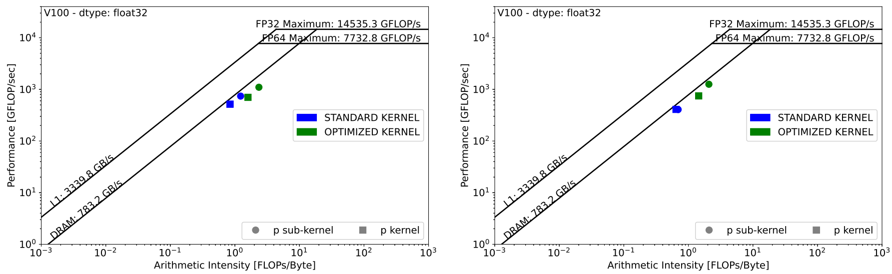
From the experiments shown in Figures 4, 5, 6, 8, 9, and 10 an analysis was performed around the functions `[CUDA memcpy HtoD]` and `[CUDA memcpy DtoH]` that handle data movement between the host and the device.

With the data of the Figure 12 was observed that this optimization was due to the increase in global L1 memory read and store transactions in *nvprof*,



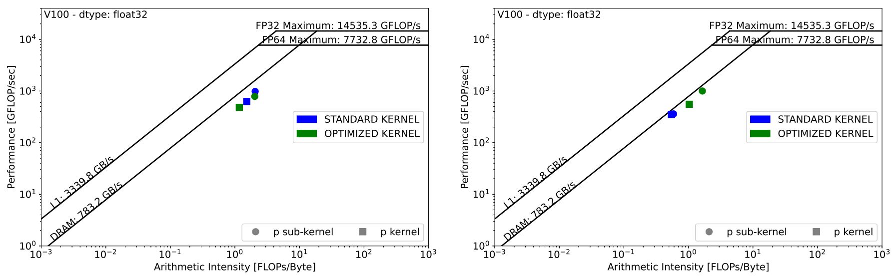
(a) (b)

Figure 8 Roofline second-order SLS viscoacoustic equation: (a) Roofline plots for a 100×100×100 model. (b) Roofline plots for a 500×500×500 model.



(a) (b)

Figure 9 Roofline second-order Kelvin-Voigt viscoacoustic equation: (a) Roofline plots for a 100×100×100 model. (b) Roofline plots for a 500×500×500 model.

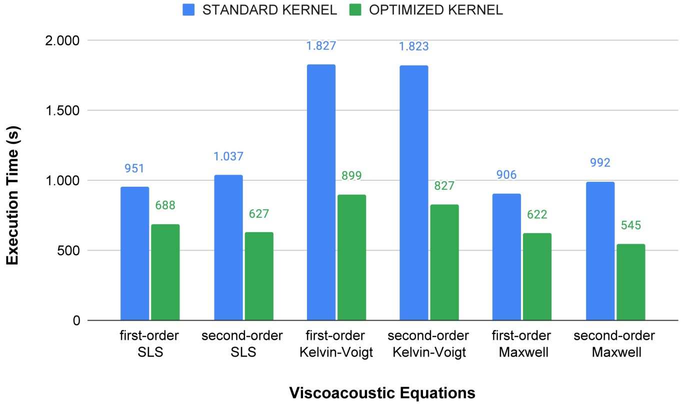


(a) (b)

Figure 10 Roofline second-order Maxwell viscoacoustic equation: (a) Roofline plots for a 100×100×100 model. (b) Roofline plots for a 500×500×500 model.

while there was a decrease in transactions of reading and storing DRAM memory in *nvprof*. It is important to emphasize that L1 memory read and store transactions are faster when compared to DRAM memory.

Another approach is to analyze the cache hit ratio. The *nvprof* tool provides information about the cache hit rate using the following metrics:



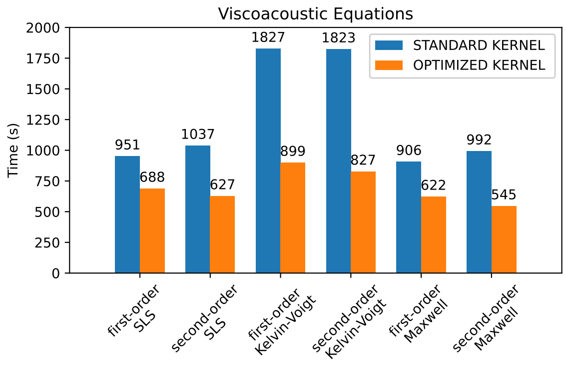


Figure 11 Forward modeling execution time for one shot.

- global_hit_rate: Hit rate for global load and store in unified L1 cache.
- local_hit_rate: Hit rate for local loads and stores.
- tex_cache_hit_rate: Unified cache hit rate.
- l2_tex_hit_rate: Hit rate at L2 cache for all requests from texture cache.

In the Figure 13, it is possible to notice that the kernels included in the `OPTIMIZED KERNEL` present higher values for global_hit_rate and tex_cache_hit_rate. In addition, there was a reduction regarding l2_tex_hit_rate.

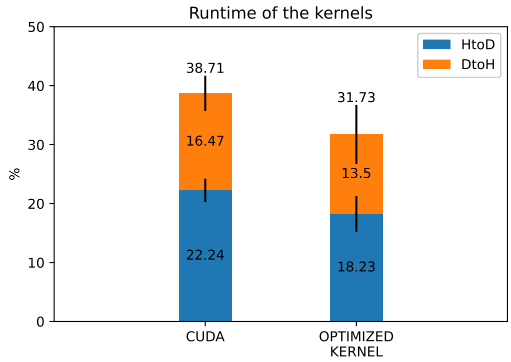


Figure 12 First-order SLS viscoacoustic equation FD kernel, Figure 4, for the `STANDARD KERNEL` the function `[CUDA memcpy HtoD]` represents 22.24% of the total runtime, while the `[CUDA memcpy DtoH]` function represents 16.47%. With the `OPTIMIZED KERNEL`, the `[CUDA memcpy HtoD]` function represents 18.23% of the total execution time, while the `[CUDA memcpy DtoH]` function represents 13.5%.

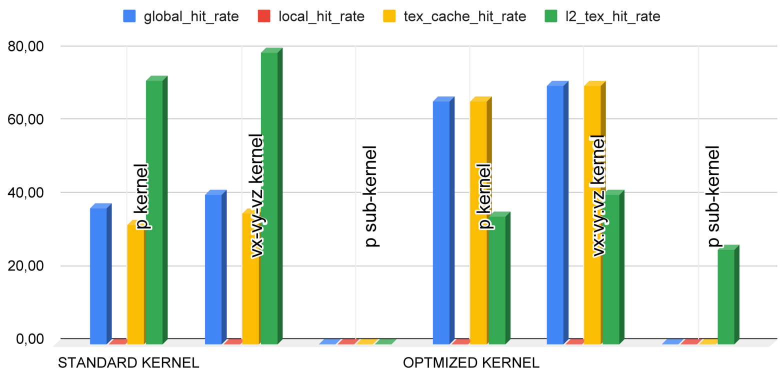


Figure 13 Application profiling: Cache hit ratio using SLS first-order viscoacoustic equation.

5 Conclusions

From the application profiling, was observed that the Kelvin-Voigt viscoacoustic equation presented greater complexity due to a more significant amount of floating-point operations performed. In addition, the Kelvin-Voigt viscoacoustic equation showed a more substantial amount of memory read and storage operations. Was noticed a more significant performance gain for the Kelvin-Voigt viscoacoustic equation, when compared to other viscoacoustic equations. This fact is explained based on the use of the *tile* clause, which provides an optimization aimed at reading and storing transactions from memory.

Was analyzed that the maximum GFLOP/s performance values obtained for the viscoacoustic wave equations FD kernels were approximately 1255

GFLOP/s. This fact shows that it is possible to increase the GFLOP/s performance since the maximum value achieved is far from the theoretical peak available in the GPU. For that, it is necessary to find more optimization alternatives. Based on the results obtained, was understood that optimizations in decreasing data movement in memory are essential to increase performance.

Some related works [1], [23], and [24] indicate strategies for optimizations that replace replicated global memory accesses with local memory accesses that substantially reduce the stencil computation execution time for any grid size. Other related works, such as [25] which aim at optimizations from the proper configuration of OpenACC directives for data management and [26] which investigates multi-GPU performance. Devito also supports distributed-memory parallelism via MPI, will also be prospected for future work the Roofline analysis of the viscoacoustic equations FD kernels execution in multi-GPU environments.

Acknowledgments

This research was executed in partnership between SENAI CIMATEC and PETROBRAS. The authors would like to acknowledge PETROLEO BRASILEIRO S.A and Agência Nacional de Petróleo, Gás Natural e Biocombustível (ANP), for the support and investments in R&D.

Declarations

5.1 Ethical approval

Not applicable.

5.2 Competing interests

Not applicable.

5.3 Authors' contributions

All authors built and reviewed the manuscript equally.

5.4 Funding

Any funding received.

5.5 Availability of data and materials

Not applicable.

References

[1] Carrijo Nasciutti, T., Panetta, J., Pais Lopes, P.: Evaluating optimizations that reduce global memory accesses of stencil computations

in GPGPUs. Concurrency and Computation: Practice and Experience **31**(18), 4929 (2019)

[2] Sano, K., Yamamoto, S., Hatsuda, Y.: Domain-specific programmable design of scalable streaming-array for power-efficient stencil computation. ACM SIGARCH Computer Architecture News **39**(4), 44–49 (2011)

[3] Said, I.: Contributions of hybrid architectures to depth imaging: a CPU, APU and GPU comparative study. PhD thesis, Université Pierre et Marie Curie-Paris VI (2015)

[4] Kukreja, N., Louboutin, M., Vieira, F., Luporini, F., Lange, M., Gorman, G.: Devito: Automated fast finite difference computation. In: 2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), pp. 11–19 (2016). IEEE

[5] Lange, M., Kukreja, N., Louboutin, M., Luporini, F., Vieira, F., Pandolfo, V., Velesko, P., Kazakas, P., Gorman, G.: Devito: Towards a generic finite difference dsl using symbolic python. In: 2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC), pp. 67–75 (2016). IEEE

[6] Louboutin, M., Luporini, F., Witte, P., Nelson, R., Bisbas, G., Thorbecke, J., Herrmann, F.J., Gorman, G.: Scaling through abstractions–high-performance vectorial wave simulations for seismic inversion with Devito. arXiv preprint arXiv:2004.10519 (2020)

[7] Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: Sympy: symbolic computing in python. PeerJ Computer Science **3**, 103 (2017). https://doi.org/10.7717/peerj-cs.103

[8] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2

[9] Konstantinidis, E., Cotronis, Y.: A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling. Journal of Parallel and Distributed Computing **107**,

37–56 (2017)

[10] Wang, Y., Yang, C., Farrell, S., Zhang, Y., Kurth, T., Williams, S.: Time-based roofline for deep learning performance analysis. In: 2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS), pp. 10–19 (2020). IEEE

[11] Yang, C., Kurth, T., Williams, S.: Hierarchical roofline analysis for gpus: Accelerating performance optimization for the nersc-9 perlmutter system. Concurrency and Computation: Practice and Experience **32**(20), 5547 (2020)

[12] Carcione, J.M.: Wave Fields in Real Media: Wave Propagation in Anisotropic, Anelastic, Porous and Electromagnetic Media, (2014)

[13] Robertsson, J.O., Blanch, J.O., Symes, W.W.: Viscoelastic finite-difference modeling. Geophysics **59**(9), 1444–1456 (1994)

[14] Carcione, J.M., Kosloff, D., Kosloff, R.: Wave propagation simulation in a linear viscoelastic medium. Geophysical Journal International **95**(3), 597–611 (1988)

[15] Dutta, G., Schuster, G.T.: Attenuation compensation for least-squares reverse time migration using the viscoacoustic-wave equation. Geophysics **79**(6), 251–262 (2014)

[16] Bai, J., Yingst, D., Bloor, R., Leveille, J.: Viscoacoustic waveform inversion of velocity structures in the time domain: Geophysics, 79. R103–R119 (2014)

[17] Jia, Z., Maggioni, M., Staiger, B., Scarpazza, D.P.: Dissecting the NVIDIA volta GPU architecture via microbenchmarking. CoRR **abs/1804.06826** (2018) https://arxiv.org/abs/1804.06826

[18] Bradley, T.: GPU performance analysis and optimization. NVIDIA Corporation (2012)

[19] Yang, C.: Hierarchical roofline analysis: How to collect data using performance tools on intel cpus and NVIDIA GPUs. arXiv preprint arXiv:2009.02449 (2020)

[20] OpenACC: Directive-based performance-portable parallel programming model for GPU Architectures. Available in: https://www.openacc.org (2021)

[21] Feki, S., Smaoui, M.: Tuning OpenACC loop execution. In: Parallel Programming with OpenACC, pp. 111–124 (2017)

[22] Rocha, R.C., Pereira, A.D., Ramos, L., Goes, L.F.: Toast: Automatic tiling for iterative stencil computations on GPUs. Concurrency and Computation: Practice and Experience **29**(8), 4053 (2017)

[23] Kim, K.-H., Kim, K.-H., Park, Q.-H.: Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model. Computer Physics Communications **182**, 1201–1207 (2011). https://doi.org/10.1016/j.cpc.2011.01.025

[24] Yang, C.: 8 steps to 3.7 TFLOP/s on NVIDIA V100 GPU: Roofline analysis and other tricks. arXiv preprint arXiv:2008.11326 (2020)

[25] Kupiainen, M., Gong, J., Axner, L., Laure, E., Nordström, J.: GPU-acceleration of a high order finite difference code using curvilinear coordinates. In: Proceedings of the 2020 International Conference on Computing, Networks and Internet of Things, pp. 41–47 (2020)

[26] Xue, W., Roy, C.J.: Multi-GPU performance optimization of a CFD code using OpenACC on different platforms. arXiv preprint arXiv:2006.02602 (2020)