Optimization of a GPU-based wave propagation application using loop tiling and loop fission

Gabriel Pinheiro da Costa[1], Peterson Nogueira Santos[1], João Henrique Speglich[1], [1]Supercomputing Center for Industrial Innovation SENAI CIMATEC (CS2I)

This paper was prepared for presentation during the 18^{th} International Congress of the Brazilian Geophysical Society held in Rio de Janeiro, Brazil, 16-19 October 2023.

Abstract

Graphics Processing Units (GPUs) possess a vast potential for parallelization that can greatly expedite the processing of massive amounts of data. Numerical methods, such as those utilized in geophysical modeling, can take advantage of this potential to achieve decreased execution times while retaining the highest levels of result quality. The present study offers an investigation of optimization techniques employed within a three-dimensional elastic model constructed using the DEVITO tool. The performance of a naive version of the model is contrasted against two optimized versions that utilize the OpenACC tile directive to enable loop tiling, working in tandem with the loop fission technique. The optimized versions were able to increase the cache hit rate and the occupancy of the GPU schedulers and reduce the execution time by 40%.

Introduction

DEVITO is a Domain-Specific Language (DSL) written in Python for stencil computation, with a focus on seismic inversion problems (Kukreja et al., 2016; Lange et al., 2016). DEVITO enables the creation of geophysical models in Python with a high degree of abstraction, thanks to the functions and classes provided by the tool. Using an internal compiler, DEVITO can translate the model written from symbolic equations in Python into a finite difference code in C/C++, which is partially optimized for execution on the specified architecture.

In addition to running on multi-node processor machines, DEVITO allows code generation targeting accelerators and GPUs. Offloading the workload to the graphics unit is done through OpenACC, which is a programming standard implemented through a library for C/C++ and Fortran, enabling code optimization in an automated way via directives (OpenACC, 2022). Among the optimizations implemented in OpenACC and available for DEVITO, the loop tiling technique stands out. This loop transformation changes the way the stencil is iterated over, where the data is no longer accessed continuously until the end of a dimension, but rather accessed in blocks with dimensions specified by the programmer (Jeffers & Reinders, 2015). The goal of this technique is to take advantage of the principle of spatial locality of data, favoring the reuse of nearby stencil points (McKinley et al., 1996; Kandemir et al., 1999).

This work focuses on the computational optimization of a DEVITO-based implementation of elastodynamic equations as described in Virieux (1986). Therefore, the objective is to develop modifications to the structure of the C++ algorithm in order to improve performance and mitigate bottlenecks without changing the nature of the equations and their solutions.

The remaining article is divided into the following sections: the Materials, Methods and Theory section describes the NSYS and NCU tools used for data collection and profiling, the forward operator of the elastic equation, the optimized approaches developed for the operator and the conditions and environment in which the tests were executed; the Results section presents and discusses a series of performance-related results collected in tests with the Naive, Tiling, and Sig Fission approaches; finally, the Conclusions section summarizes the main conclusions that can be drawn from the presented results.

Materials, Methods and Theory

Profiling Tools

Nsight Systems (NSYS) is a GPU profiling tool that provides a comprehensive view of the system throughout the application's life cycle. It can track calls to external APIs, system operations, and the use of other hardware such as the CPU, as well as record bilateral memory traffic between the host and device (Nvidia, 2023b). In the present study NSYS was used to identify which kernels were responsible for most of the computational effort involved in the application.

Nsight Compute (NCU) is a CUDA kernel profiler that provides a range of metrics for data collection and analysis. By conducting a vertical analysis restricted exclusively to the analyzed kernel, it is the recommended tool for obtaining statistically and isolated information, and for performing performance comparisons between two or more versions of the same kernel (Nvidia, 2023a).

The elastic wave equation

The propagation of seismic waves in heterogeneous, isotropic, elastic earth media can be expressed by the elastodynamic equations (Virieux, 1986)

$$\begin{cases} \rho \dfrac{\partial v_x}{\partial t} - \left(\dfrac{\partial \sigma_{xx}}{\partial x} + \dfrac{\partial \sigma_{xy}}{\partial y} + \dfrac{\partial \sigma_{xz}}{\partial z} \right) = 0, \\ \rho \dfrac{\partial v_y}{\partial t} - \left(\dfrac{\partial \sigma_{yx}}{\partial x} + \dfrac{\partial \sigma_{yy}}{\partial y} + \dfrac{\partial \sigma_{yz}}{\partial z} \right) = 0, \\ \rho \dfrac{\partial v_z}{\partial t} - \left(\dfrac{\partial \sigma_{zx}}{\partial x} + \dfrac{\partial \sigma_{zy}}{\partial y} + \dfrac{\partial \sigma_{zz}}{\partial z} \right) = 0, \\ \dfrac{\partial \sigma_{xx}}{\partial t} - (\lambda + 2\mu) \dfrac{\partial v_x}{\partial x} - \lambda \left(\dfrac{\partial v_y}{\partial y} + \dfrac{\partial v_z}{\partial z} \right) = f_{xx}, \\ \dfrac{\partial \sigma_{yy}}{\partial t} - (\lambda + 2\mu) \dfrac{\partial v_y}{\partial y} - \lambda \left(\dfrac{\partial v_x}{\partial x} + \dfrac{\partial v_z}{\partial z} \right) = f_{yy}, \\ \dfrac{\partial \sigma_{zz}}{\partial t} - (\lambda + 2\mu) \dfrac{\partial v_z}{\partial z} - \lambda \left(\dfrac{\partial v_x}{\partial x} + \dfrac{\partial v_y}{\partial y} \right) = f_{zz}, \\ \dfrac{\partial \sigma_{yz}}{\partial t} - \mu \left(\dfrac{\partial v_z}{\partial y} + \dfrac{\partial v_y}{\partial z} \right) = 0, \\ \dfrac{\partial \sigma_{zx}}{\partial t} - \mu \left(\dfrac{\partial v_x}{\partial z} + \dfrac{\partial v_z}{\partial x} \right) = 0, \\ \dfrac{\partial \sigma_{xy}}{\partial t} - \mu \left(\dfrac{\partial v_x}{\partial y} + \dfrac{\partial v_y}{\partial x} \right) = 0. \end{cases} \quad (1)$$

where (v_x, v_y, v_z) are the horizontal and vertical particle velocity fields, $(\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{yz}, \sigma_{zx}, \sigma_{xy})$ are the stress fields, f_{xx}, f_{yy}, f_{zz} are the source terms, ρ is density, λ and μ are the Lamé parameters. These coefficients describe the spatially variable property of the earth. The latter is related to the seismic P- and S-wave velocities via $\lambda + 2\mu = \rho V_p^2$ and $\mu = \rho V_s^2$.

Approaches

Analyses performed with the NSYS profiler on the generated C++ code showed that 98.3% of the computational effort employed in the total execution of the application is concentrated in just two kernels: V (53.2%) and Sig (45.1%). These kernels are responsible for calculating the particle velocity fields (V) and the stress fields (Sig), as shown in Equation 1. Therefore, optimization efforts focused on extracting performance from these two kernels. The present work compares the results obtained from the standard version of the operator generated by DEVITO (without any type of tuning or modification) with two new approaches developed for performance extraction, as described below:

- Naive: the DEVITO default approach, in which the main optimization is the use of the ”collapse” clause to transform a three-dimensional nesting into a one-dimensional iteration loop.
- Tiling: an approach that replaces the ”collapse” clause by the ”tile” clause in the V and Sig kernels, with tuned values. The V kernel uses tiles with dimensions of (32, 4, 4), while the Sig kernel uses dimensions of (16, 4, 4).
- Sig Fission: an approach that combines the techniques of loop tiling and loop fission. In this approach, the V kernel maintains the tile dimensions of (32, 4, 4). The loop fission technique divides the workload of a nesting into two or more nests of smaller workload but with the same iteration space (Cardoso et al., 2017), and it was applied to the Sig kernel by separating the cross-derivatives into one nest, and the derivatives taken in the same dimension into another nest. The ”tile” clause is applied again to these two new nests, maintaining the (32, 4, 4) dimension applied in the V nest.

The tile dimensions of each of the kernels were determined empirically, always choosing the ones that achieved the best performance. Algorithms 1, 2, and 3 illustrate the pseudocodes of the Naive, Tiling, and Sig Fission approaches, respectively

Algorithm 1 Naive Approach

```
1:  procedure COMPUTE V
2:      pragma acc parallel loop collapse(3) ...
3:      for x... do
4:          for y... do
5:              for z... do
6:                  Compute vx, vy, vz
7:              end for
8:          end for
9:      end for
10: end procedure
11: ...
12: procedure COMPUTE SIG
13:     pragma acc parallel loop collapse(3) ...
14:     for x... do
15:         for y... do
16:             for z... do
17:                 Compute σxx, σyy, σzz, σyz, σzx, σxy
18:             end for
19:         end for
20:     end for
21: end procedure
```

Algorithm 2 Tiling Approach

```
1: procedure COMPUTE V
2:     pragma acc parallel loop tile(32,4,4) ...
3:     for x... do
4:         for y... do
5:             for z... do
6:                 Compute vx, vy and vz
7:             end for
8:         end for
9:     end for
10: end procedure
11: ...
12: procedure COMPUTE SIG
13:     pragma acc parallel loop tile(16,4,4) ...
14:     for x... do
15:         for y... do
16:             for z... do
17:                 Compute σxx, σyy, σzz, σyz, σzx, σxy
18:             end for
19:         end for
20:     end for
21: end procedure
```

Algorithm 3 Sig Fission Approach

```
1: procedure COMPUTE V
2:     pragma acc parallel loop tile(32,4,4) ...
3:     for x... do
4:         for y... do
5:             for z... do
6:                 Compute vx, vy, vz
7:             end for
8:         end for
9:     end for
10: end procedure
11: ...
12: procedure COMPUTE SIG SAME DIM.
13:     pragma acc parallel loop tile(32,4,4) ...
14:     for x... do
15:         for y... do
16:             for z... do
17:                 Compute σxx, σyy, σzz
18:             end for
19:         end for
20:     end for
21: end procedure
22: ...
23: procedure COMPUTE SIG CROSSED
24:     pragma acc parallel loop tile(32,4,4) ...
25:     for x... do
26:         for y... do
27:             for z... do
28:                 Compute σyz, σzx, σxy
29:             end for
30:         end for
31:     end for
32: end procedure
```

Environment and parameters

All tests were performed on a NVIDIA Tesla V100 SXM2 32 GB card, with dedicated access to the hardware, eliminating any potential interference from other applications. The runtime results presented in this study are averaged over five runs conducted under identical conditions and parameters.

The simulations were performed on a three-dimensional model with 400 elements in each dimension and simulation time of 1000 milliseconds. A more detailed overview of the execution parameters can be found in the Table 1 below.

Parameter	Description	Value
nx	Number of points in X	400
ny	Number of points in Y	400
nz	Number of points in Z	400
tn	Simulation time (ms)	1000
dt	Discretization timestep size	1.414
so	Space order	16

Table 1 – Simulation Parameters

It is important to emphasize that the integrity of the results obtained by all approaches was rigorously checked, and the quality of the results was not compromised.

Results

Table 2 presents the average execution times in seconds for the three approaches and the respective saved time percentage of the optimized approaches compared to the time obtained by the Naive version. The number of cycles required to execute the two main kernels in each of the approaches is shown in Table 3. For better visualization and ease of comparison, all data collected from the two additional kernels generated in Sig Fission will be presented in a combined manner, through a weighted average (taken according to the workload of the kernels) as if they continued to be a single kernel.

The results shown in both tables reveal that the optimized approaches were able to reduce the number of cycles required for processing the two kernels, and consequently significantly reduce the execution time of the application.

Approach	Time (s)	Saved (%)
Naive	85.87	-
Tiling	58.48	33
Sig Fission	52.22	40

Table 2 – Elapsed time and saved time percentage of all approaches.

Tables 4 and 5 present the computational performance and arithmetic intensity of each of the approaches in the V and Sig kernels, respectively. Computational performance is a metric that measures the number of floating-point operations per second (FLOP/s), and arithmetic intensity

Approach	V Cycles	Sig Cycles
Naive	56,741,131	67,458,321
Tiling	31,355,925	51,141,142
Sig Fission	31,289,283	41,885,863

Table 3 – Cycles spent in each kernel.

is the number of floating-point operations per byte (FLOP/byte).

Compared to the Naive version, the application of tiling in the V kernel in the two optimized approaches is able to increase both computational performance and arithmetic intensity, which exceeds twice the value obtained by the Naive approach. In the Sig kernel, the two optimized approaches allow for performance gain in both metrics analyzed. Both the Tiling and Sig Fission versions come close to doubling the arithmetic intensity values obtained by the Naive version, and also show substantial improvements in performance, with the Sig Fission approach achieving the best results.

Approach	Perf. (FLOP/s)	Arit. Int. (FLOP/byte)
Naive	$0.469 \cdot 10^{12}$	0.72
Tiling	$0.834 \cdot 10^{12}$	1.63
Sig Fission	$0.834 \cdot 10^{12}$	1.63

Table 4 – Performance and Arithmetic Intensity on kernel V.

Approach	Perf. (FLOP/s)	Arit. Int. (FLOP/byte)
Naive	$0.618 \cdot 10^{12}$	1.07
Tiling	$0.799 \cdot 10^{12}$	2.10
Sig Fission	$0.965 \cdot 10^{12}$	1.99

Table 5 – Performance and Arithmetic Intensity on kernel Sig.

For the V kernel, the L1 cache hit rate has increased with the application of loop tiling in the Tiling and Sig Fission approaches, and the L2 hit rate in both optimized approaches remained very close to that obtained by the Naive version, as shown in the graph of Figure 1. The graph in Figure 2 shows that the Sig kernel presented improvements in L1 cache hit rates, both with the isolated application of loop tiling in the Tiling approach and with loop fission combined with tiling in the Sig Fission approach. In L2, the Naive version obtained the highest hit rate in this kernel, but with very little difference compared to the two optimized versions.

The values obtained in this section by the two optimized approaches, in both kernels, converge with the results presented previously. The increase in cache hit rates allows for a more efficient use of data, reducing processing bottlenecks and allowing the application to make better use of the available processing power (increasing FLOP/s performance and arithmetic intensity).

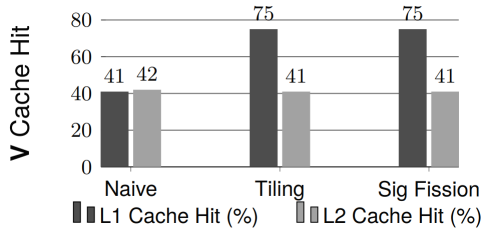

***Figure** 1 – L1 and L2 cache hit rates on kernel V.*

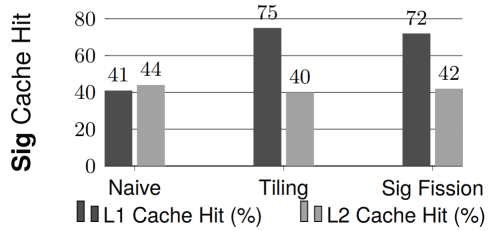

***Figure** 2 – L1 and L2 cache hit rates on kernel Sig.*

Other interesting metrics for performance analysis of an application are the warp issue rate and the number of registers required for a thread to issue an instruction. A GPU has multiple microprocessors called Stream Multiprocessors (SM), and each SM has a set of schedulers responsible for managing and issuing groups of threads carrying instructions. These groups of threads are called warps. The warp issue rate, therefore, refers to the average number of warps issued per scheduler in a certain number of cycles. Nevertheless, each thread requires a certain amount of hardware registers to issue an instruction, so the heavier the workload, the more registers will be required, leading to increased competition for available hardware resources, which can reduce the scheduler issue rate.

Figure 3 shows that the use of loop tiling in the Tiling and Sig Fission versions for the V kernel was able to substantially increase the warp emission rate in this fraction of the operator, surpassing twice the value obtained in the Naive version. The number of registers required for a warp to be issued also showed a similar behavior, dropping to just over half with the application of loop tiling in the two developed approaches.

In the Sig kernel, the Tiling approach was able to slightly increase the average number of issued warps compared to the Naive approach. In contrast, the application of loop tiling after the fission process in the Sig Fission approach achieved a significant increase in the issued rate, which proves that the fission process led to an optimization in the use of schedulers. The application of loop tiling only in the Sig kernel used in the Tiling approach showed minimal variations compared to the Naive version, as shown in the graph in Figure 4. However, the Sig Fission approach was able to reduce the need for registers by almost half.

The decrease in the number of registers required per thread, observed in the Tiling and Sig Fission approaches

for kernel V, and in the Sig Fission approach for kernel Sig, is one of the factors that allows an increase in the occupancy of the SM's schedulers.

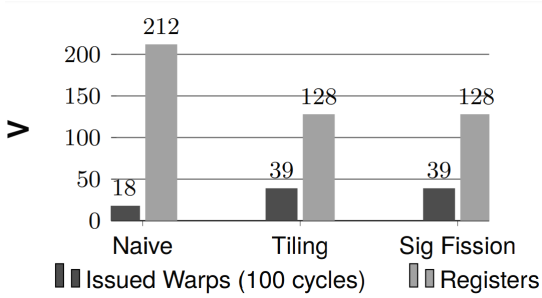

***Figure** 3 – Rate of Issued warps every 100 cycles and amount of registers needed on kernel V.*

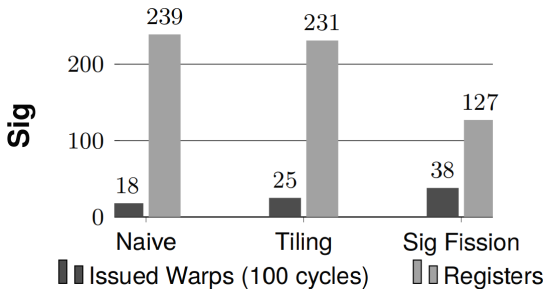

***Figure** 4 – Rate of Issued warps every 100 cycles and amount of registers needed on kernel Sig.*

The main advantage of the two optimized approaches in the V kernel was the reduction in the cache miss rate, allowing a more efficient use of data and a decrease in the need for expensive memory accesses, since the necessary data for processing are now more frequently found in the cache memory. The reduction of waiting bottlenecks caused by accesses to main memory allowed for a better use of available computational resources, increasing performance and arithmetic intensity. These gains explain the superior performance of the Tiling and Sig Fission approaches in this kernel, if compared to the Naive approach, which obtained significantly lower cache hit rates.

In the Sig kernel, the reduction in cache miss rate is also the main factor responsible for the positive results of the Tiling and Sig Fission approaches. Once again, the more efficient use of data and the reduction in the need for costly memory accesses allowed for better utilization of the hardware computational power. However, the Sig Fission approach proved to be more efficient in reducing the number of registers required per thread, which led to a higher rate of issued warps. This higher emission of warps also contributes to an even better utilization of the processing power offered by the hardware, making the application of loop fission deliver superior results, even though the two optimized approaches achieved very similar cache hit rates in both levels of cache.

Conclusions

Based on the results collected by the NCU profiler, as well as the average execution times obtained, it is possible to affirm that the two developed approaches were able to overcome some of the limitations of the Naive approach and deliver positive results. Both loop transformation techniques used had positive impacts on the application, with loop tiling being able to take advantage of the principle of data locality and making better use of cache memory, and loop fission being able to increase computational performance in the Sig kernel.

The Sig Fission approach is the one that achieves the best results by combining the strategies that obtained the most expressive results in the two main kernels of the operator: using loop tiling of dimensions (32, 4, 4) in the V kernel, and using loop fission separating the cross derivatives followed by the use of loop tiling of dimensions (32, 4, 4) in the Sig kernels.

Acknowledgments

This work was developed in partnership between SENAI CIMATEC and PETROBRAS. The authors acknowledge PETROLEO BRASILEIRO S.A and the Agência Nacional de Petróleo, Gás Natural e Biocombustível (ANP), for their support and investment in research and development.

References

Cardoso, J. M. P., de Figueired Coutinho, J. G. & Diniz, P. C., 2017. Embedded computing for high performance: Efficient mapping of computations using customization, code transformations and compilation, Morgan Kaufmann, 137–183.

Jeffers, J. & Reinders, J., 2015. High performance parallelism pearls volume two: multicore and many-core programming approaches, Morgan Kaufmann.

Kandemir, M., Ramanujam, J. & Choudhary, A., 1999. Improving cache locality by a combination of loop and data transformations, IEEE Transactions on Computers, vol. 48(2): 159–167.

Kukreja, N. et al., 2016. Devito: Automated fast finite difference computation, in: 2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), IEEE, 11–19.

Lange, M. et al., 2016. Devito: Towards a generic finite difference dsl using symbolic python, in: 2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC), IEEE, 67–75.

McKinley, K. S., Carr, S. & Tseng, C.-W., 1996. Improving data locality with loop transformations, ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 18(4): 424–453.

Nvidia, 2023a. Nsight Compute: Developer Tools Documantion., https://docs.nvidia.com/nsight-compute/NsightCompute/index.html.

Nvidia, 2023b. Nsight Systems: Developer Tools Documantion., https://docs.nvidia.com/nsight-systems/UserGuide/index.html.

OpenACC, 2022. OpenACC Programming and Best Practices Guide, https://www.openacc.org/sites/default/files/inline-files/openacc-guide.pdf.

Virieux, J., 1986. P-sv wave propagation in heterogeneous media: Velocity-stress finite-difference method, Geophysics, vol. 51(4): 889–901.