Optimization of a Geophysical Application in GPU Through the Loop Tiling Technique

Gabriel Pinheiro da Costa[1*], Murilo Boratto[2], Marcelo Oliveira da Silva[3], João Henrique Speglich[4]
[1]Supercomputing Center for Industrial Innovation SENAI CIMATEC (CS2I); Salvador, Bahia, Brazil

This work aims to present the results obtained in optimizing a viscoacoustic geophysical model written with the DEVITO tool and optimized using the OpenACC tile directive for GPU execution. We compared three versions of the operator using the NVIDIA NCU profiling tool: Naive, Tiling (32,4,4), and Mixed Tiling. The Naive version does not use the loop tiling technique, the Tiling (32,4,4) version applies a tile of dimensions (32, 4, 4), and the Mixed Tiling version uses different tile sizes to other loop nests. Analyzing the experimental results, it is notable that the optimized versions substantially increase the cache hit rates and reduce the execution time by about 50%, attesting to the validity of the proposed solutions.
Keywords: HPC. Optimization. OpenACC. Loop Tiling. DEVITO.

Introduction

DEVITO is a tool for implementing computational mesh models in symbolic language. It is a Python package with automated code generation that allows portability to different platforms [1]. DEVITO is a helpful tool for building geophysical models for parallel architectures.

DEVITO allows OpenACC to offload the workload to a device with more processing power, such as a GPU. OpenACC is a programming standard for optimizing C, C++, and Fortran code. The user uses directives to inform the regions of the code that he wants to optimize in an automated way [2].

Through environmental variables, DEVITO can generate code with OpenACC directives capable of promoting GPU execution and parallelization. One such directive is the tile directive, which applies the loop tiling technique [3] in a loop nest with the dimensions defined as a parameter. The loop tiling technique modifies a loop nest, so data is no longer accessed sequentially in one dimension but in multidimensional blocks of predefined size [3]. This transformation uses better nests' spatial and temporal locality [4,5].

Received on 18 December 2022; revised 5 January 2023.
Address for correspondence: Gabriel Pinheiro da Costa. Avenida Orlando Gomes, No.1845, Piatã, Salvador, Bahia, Brazil. Zipcode: 41650-010. E-mail: gabriel.pinheiro@fieb.org.br.

J Bioeng. Tech. Health 2023;6(Suppl.1):1-5.

Materials and Methods

Developed Approaches

In the analyzed application, two kernels have a more extensive workload, responsible for a large part of the required computational effort: R and RP.

Thus, we selected three approaches for analysis in the viscoacoustic model.

Naive

The tool's default approach, without any parameter optimization or modification of the generated code. It only counts on the "advanced" default optimization level. It is the most straightforward approach.

Tiling (32,4,4)

The approach only uses the part-tile flag (DEVITO native), not requiring any transformations of the .cpp code generated by the framework. This flag applies the loop tiling technique to all loop nests restrained in the operator, using OpenACC's tile directive. The combinations of dimensions that achieved the best performance were 32 elements in x, 4 elements in y, and 4 in z [simply: (32,4,4)].

Mixed Tiling

It works with different tile sizes for loop nests.

Using the OpenACC tile directive, the R kernel applies the technique of loop tiling with dimensions (32,8,4), whereas the RP kernel, through the same process, applies a tiling of sizes (32,4,4). This variation in tile dimensions in the two kernels occurs because the R kernel reached the best performance with measurements (32,8,4), while the RP kernel got its peak performance with dimensions (32,4,4).

Hardware and Experiments

NVIDIA Tesla V100 SXM2 32 GB cards performed all tests, with exclusive access to the hardware and no competition with other applications. The following environmental variables were used to enable GPU execution with OpenACC:

- DEVITO_PLATFORM=nvidiaX;
- DEVITO_ARCH=nvc;
- DEVITO_LANGUAGE=openacc.

The tests were carried out on a three-dimensional model with 701 elements in each dimension, a value that pushed the GPU memory storage capacity to the limit. Each run conducted for 1,000 iterations and applied a space order of 16 elements. All runtime results are means of three runs performed under the same conditions and parameters.

Profiling Tool - Nsight Compute (NCU)

Nsight Compute (NCU) [6] is a CUDA kernel profiler that has a graphical interface and operates by the command line. It offers a series of metrics and sections (metric grouping), which can be collected in a customized way by the user to restrict the scope of the analysis. It is the correct tool to obtain statistical and mathematical information for each application's kernel. Three sections of the NCU were used to analyze the kernels presented in this work.

GPU Speed Of Light Roofline Chart

This section brings two metrics of great value for performance analysis. Arithmetic intensity is the ratio of floating-point operations performed per second, memory transfer in bytes, and per second. This is a metric strictly related to memory traffic. Performance measures the number of floating point operations per second (FLOP/s) and indicates computational performance.

Memory Workload Analysis

Displays data-related GPU memory resources, including cache hit rates. The most relevant metrics in this section are cache hit rates on L1 and L2.

Scheduler Statistics

This section summarizes the schedulers that issue instructions. Each scheduler maintains a group of warps from which it can pull instructions. In each cycle, each scheduler checks the status of the warps allocated in its group (Active Warps), looking for warps that are not stalled (Eligible Warps) and, therefore, ready to issue its next instruction. An eligible warp is then selected, and its instructions are issued (Issued Warp). The parameter that strongly impacts the occupancy rate of a scheduler is the number of registers needed per thread. Each GPU SM has 4 sub-partitions, each one with a scheduler.

Results and Discussion

Table 1 compares the execution times in seconds for each of the three approaches. Table 2 presents the number of cycles spent on each one of the kernels.

Table 3 shows the R kernel results for this NCU GPU Speed of Light Roofline Chart section. A significant improvement in both performance and arithmetic intensity is presented in both optimized approaches. The Tiling (32,4,4) approach almost tripled the values of these two metrics concerning that obtained by the Naive version, and the Mixed Tiling approach was able to surpass three times the values obtained by the Naive version in both metrics. In the RP kernel, the improvement was also noticeable (Table 4), reaching an average of

FLOP per byte almost twice that presented by the Naive version in the two optimized approaches. The average performance also increased notably in the Tiling (32,4,4) and Mixed Tiling approaches. The results presented by the two tables indicate an increase in processing capacity over the same volume of data (arithmetic intensity) and better use of available computational resources (performance).

Tables 5 and 6 show the results of the R and RP kernels for the Memory Workload Analysis section for the three approaches. We significantly increased the cache rates achieved in L1 and L2 in the R kernel with the two optimized approaches. The Mixed Tiling approach performed better in L1, increasing the hit rate at this cache level by almost 44 percentage points compared to the Naive version and by more than 6 percentage points compared to the Tiling (32,4,4) version. In the RP kernel, the optimized approach also increased the cache hit rate at both levels. In L1, this increase is more significant, getting close to reaching twice that obtained by the Naive version, while in L2, the growth is lower but still perceptible, rising by about 7 percentage points compared to the non-optimized version. The values obtained in this section by the two optimized approaches in both kernels converge with the results presented. Increasing cache hit rates allows more efficient use of data, reducing processing bottlenecks and allowing the application to use better available processing power (improved performance and arithmetic intensity). The higher L1 cache hit rate of the Mixed Tiling approach compared to the Tiling (32,4,4) approach is one factor that explains the slightly superior performance of one strategy over the other.

Table 1. Execution times.

Approach	Time (s)
Naive	220.14
Tiling (32,4,4)	112.47
Mixed Tiling	110.01

Table 2. Cycles spent on each kernel.

Approach	R	RP
Naive	117.272.997	276.679.918
Tiling (32,4,4)	52.049.206	110.608.921
Mixed Tiling	43.834.245	110.124.618

Table 3. R - GPU speed of light roofline chart.

Approach	Performance (FLOP/s)	Arithmetic Intensity (FLOP/byte)
Naive	$0.317 \cdot 10^{12}$	0.53
Tiling (32,4,4)	$0.953 \cdot 10^{12}$	1.57
Mixed Tiling	$1.069 \cdot 10^{12}$	1.84

Table 4. RP - GPU speed of light roofline chart.

Approach	Performance (FLOP/s)	Arithmetic Intensity (FLOP/byte)
Naive	$0.370 \cdot 10^{12}$	0.63
Tiling (32,4,4)	$0.581 \cdot 10^{12}$	1.23
Mixed Tiling	$0.583 \cdot 10^{12}$	1.22

Table 5. R - Memory Workload Analysis.

Approach	L1 Cache Hit (%)	L2 Cache Hit (%)
Naive	34.37	36.38
Tiling (32,4,4)	71.42	59.95
Mixed Tiling	77.73	58.57

Table 6. RP - Memory Workload Analysis.

Approach	L1 Cache Hit (%)	L2 Cache Hit (%)
Naive	33.68	30.61
Tiling (32,4,4)	61.60	37.92
Mixed Tiling	61.59	37.54

Table 7 presents the R kernel results for the scheduler statistics section. Although there is no significant variation in the theoretical maximum amount of warps per scheduler between the Tiling(32,4,4) and Naive versions, the rates of eligible and effectively issued warps are notably accentuated in the Tiling (32,4,4) and Mixed Tiling, which reach values that exceed three times that obtained by Naive, with the Mixed Tiling approach having slightly higher values. A similar but more timid result is obtained by the optimized approaches in the RP kernel, as shown in Table 8. The theoretical maximum of warps per scheduler does not change; however, the amount of eligible warps exceeds twice the Naive version, and the average warps emitted per cycle jumps from 0.14 to 0.25. The results point to better use of the schedulers in the Tiling(32,4,4) and Mixed Tiling versions, which start to emit more warps per cycle and mitigate the possibilities of taking the computational resources to idleness. The increase in cache hit rates in the optimized approaches is the main factor that increased the average warps emitted.

Conclusion

The results reveal that both kernels are positively sensitive to loop tiling. In the R kernel, the application of the OpenACC tiling directive had the main effect of substantially increasing the cache hit rates at both levels in the two optimized approaches. This better use of cache memory allowed an increase in computational efficiency, observed in improving metrics such as arithmetic intensity, performance, and the rate of warps emitted per cycle. After applying the loop tiling technique, the RP kernel went through a process similar to that of the R kernel, which had as its main positive effect the increase in cache hit rates. This more efficient use of cache memory increased computational efficiency, increasing metrics such as arithmetic intensity, performance, and the rate of warps emitted. Therefore, despite the very similar version of the two optimized approaches, the slightly higher cache hit rate of the Mixed Tiling approach over Tiling(32,4,4) in L1 ends up giving the process that uses mixed tiling dimensions a slightly better performance.

Table 7. R - Scheduler statistics.

Approach	Theoretical Maximum	Eligible	Emitted
Naive	5	0.19	0.11
Tiling (32,4,4)	4	0.65	0.35
Mixed Tiling	8	1.16	0.42

Table 8. RP - Scheduler statistics.

Approach	Theoretical Maximum	Eligible	Emitted
Naive	4	0.19	0.14
Tiling (32,4,4)	4	0.40	0.25
Mixed Tiling	4	0.40	0.25

References

1. Devito: Symbolic Finite Difference Computation. Available at: https://www.devitoproject.org. Accessed Nov. 2022.
2. OpenACC Programming and Best Practices Guide. Available at: openacc.org. Accessed on: 20 Nov. 2022.
3. Jeffers J, Reinders J. High-performance parallelism pearls volume two: multicore and many-core programming approaches. Morgan Kaufmann 2015:410-416.
4. McKinley KS, Carr S, Tseng C-W. Improving data locality with loop transformations. ACM Transactions on Programming Languages and Systems (TOPLAS) 1996;18(4):424-453.
5. Kandemir M, Ramanujam J, Choudhary A. Improving cache locality by a combination of loop and data transformations. IEEE Transactions on Computers 1999;48(2):159-167.
6. Nsight Compute: Developer Tools Documentation. Available at: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html. Accessed Nov. 2022.