Seismic Wave Stencil Computation Performance Comparison

Otávio F. Freitas[1], **Lucas Cilento**[3],
Renato S. Guimarães[1], **Jaime F. Souza**[2], **Hermes Senger**[2], **Edson S. Gomi**[1]

[1]Escola Politécnica – Universidade de São Paulo (USP), São Paulo – SP – Brazil

[2]Departamento de Computação – Universidade Federal de São Carlos, São Carlos – SP – Brazil

[3]Escola Superior de Propaganda e Marketing (ESPM), São Paulo – SP – Brazil

Abstract. *This work aims to provide comparative results between OpenCL and oneAPI performance on FPGAs, GPUs and CPUs regarding the execution of stencils in the simulation of seismic acoustic wave propagation. Our results show that the naive stencil code performs better on GPU and CPU than FPGA without strong knowledge on how to implement optimizations. Another result is that oneAPI code is easier and faster to implement than OpenCL as it automatically applies some FPGA-specific optimizations at compile time.*

1. Introduction

The concept of a hardware accelerator is not new. However, its popularity has grown a lot over the last years, primarily due to the end of Moore's Law, which famously predicted that the computation capacity would double every couple of years, which was almost precisely correct for close to five decades. However, this prediction is starting to slow down as we are likely reaching the physical limits of possible transistor size. However, humanity almost counted on this increase, and it is sudden stop caught whole sectors off-guard, directing them to search for hardware architectures alternatives to continue to gain speed. FPGAs are convenient configurable devices to implement new ideas quickly and have been used for a while now. However, the design process of complex hardware using low-level hardware description languages, like VHDL and Verilog, is quite tricky. However, in recent years we got multiple platform-heterogeneous languages to make the design process more manageable. In this work, our primary goal is to analyze the difference in performance between implementations using languages like C, OpenCL, and oneAPI, in different devices (CPU, GPU, and FPGA). This work is part of a more significant research project regarding seismic computation (forward wave propagation and seismic inversion), so we use the wave propagation simulation algorithm as the benchmark.

2. OpenCL and oneAPI

OpenCL (Open Computing Language) is an open and cross-industry standard for parallel computation in several hardware accelerators based on CPUs, GPUs, and FPGAs. The standard is maintained by Khronos Group[1], a non-profit consortium formed by leading companies in the high-performance computing field. The OpenCL framework comprises several tools (compiler, device drivers, and hardware information) whose purpose is to enable non-experts in hardware design to develop accelerators for high-performance computing applications. Using OpenCL, the designer describes the accelerator using a C or

[1]https://www.khronos.org/opencl/

C++-based language. In this work, we used Intel and NVIDIA's OpenCL compilers for FPGA and GPU implementations, respectively. Intel oneAPI[2] is another open standard for accelerator architectures. Its primary language is the Data Parallel C++ (DPC++), which is a language based on C++17 core language. DPC++ implements the SYCL specification from The Kronos Group. The structure of both OpenCL and oneAPI have the host and device parts. The host code usually runs on a CPU, and the kernel code is intended to be executed on a accelerator device (GPU or FPGA).

3. Stencil for Seismic Wave Propagation

In seismic surveys, geophones record reflected waves generated by acoustic sources. These recorded signals are the primary information to infer the subsurface structure using seismic inversion computation algorithms. A preferred seismic inversion method is the Full Waveform Inversion (FWI). However, FWI is computationally expensive, taking a long time to compute the thousands of runs necessary to achieve the required precision in the subsurface structure identification. The most expensive part of this computation is the simulation of the forward wave propagation. In the 2D case, the equation that describes the wave propagation as follows:

$$\frac{\partial^2 p(x,z,t)}{\partial t^2} = c(x,z)^2 [\frac{\partial^2 p(x,z,t)}{\partial x^2} + \frac{\partial^2 p(x,z,t)}{\partial y^2}] \tag{1}$$

To solve the acoustic wave equation, we use the Finite Difference method, working with a second-order discretization using a four-point 2D stencil for the second derivative:

$$p_{j,k}^{n+1} = dt^2 c_{j,k}^2 [\frac{p_{j+1,k}^n - 2p_{j,k}^n + p_{j-1,k}^n}{dx^2} + \frac{p_{j,k+1}^n - 2p_{j,k}^n + p_{j,k-1}^n}{dy^2}] + 2p_{j,k}^n - p_{j,k}^{n-1} \tag{2}$$

In our discretized equation the variable p represents the wave pressure field, C is the wave velocity, n is the time, and j and k represent indexes of each axis of our two-dimensional grid (x, z). We can vary the size of those dimensions to simulate larger physical areas. The grid size will later become a parameter called grid size in our algorithm. The second parameter that we have is the number of iterations (or time steps), represented as *n*, which defines the time dimension in the equation 2. The numerical solution of the equation implements a stencil code, which is a memory-intensive pattern [Justin Holewinski and Sadayappan 2012, Schäfer and Fey 2011]. Stencil computations involve updating values associated with points on a regular grid over time stepping. As illustrated in Figure 1, we calculate the value of a certain grid point in the next time step from its previous value and the values at a set of neighboring points. In our 2D grid, we implement a four-point neighborhood, which considers only direct neighbors along each axis (cardinal directions).

4. Method

In order to attempt to make a more plural performance comparison, we wrote discretized wave propagation algorithms in two distinct platform-heterogeneous frame-

[2]https://www.oneapi.io/

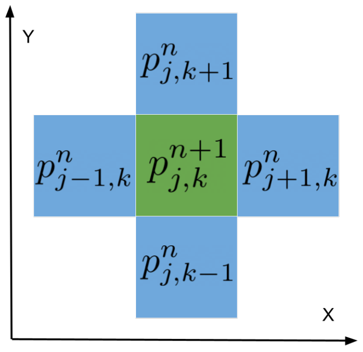

Figure 1. Representation of the four-point 2D stencil.

works: OpenCL and Intel oneAPI. We then executed those algorithms in three different platforms: a CPU (Intel Xeon Gold 6130), a GPU (NVIDIA Tesla V100), and an FPGA (Intel Arria 10). All of our algorithm versions had the possibility of changing the parameters, cited in section 3, before compilation. We collected the OpenCL and oneAPI kernel execution time of every possible parameter combination of the following: 128, 256, 512, and 1024 for grid sizes (we always used a square matrix); and 10, 50, 100, and 200 for the number of iterations, when varying one parameter we kept the other one at it's maximum value. As another control measure, we also ran every kernel twenty times for each parameter combination to minimize variance. Moreover, whenever we cite an execution time of any given parameter configuration, we refer to the average time of those executions, which all have less than 1% of standard deviation.

5. Results

As we can see in the leftmost plot of Figure 2, the execution time increases almost linearly with an increased number of iterations, as expected. We can also observe that (1) the CPU implementation in C language performed better than our GPU code in OpenCL, and (2) the oneAPI implementation has a significantly good advantage over OpenCL in FPGA. The rightmost plot of Figure 2 shows that the computation time increases exponentially with the grid size.

6. Discussion

Initially, we expected that FPGA performance would match the one provided by GPUs, but the OpenCL implementation performance on GPU was significantly better than our implementation based on OpenCL and Intel oneAPI for FPGA. The goal, however, was not to compare devices' performance but to assess portability and easiness of implementation using code with slight changes. Our hypothesis for the disparity between OpenCL and oneAPI computing times in FPGAs is that the initial code for CPUs was written in C language, with OpenMP, and only the kernel was ported for OpenCL with not enough FPGA specific optimizations implemented to assure a competitive performance. This fact made the results in OpenCL's times being exponentially worst. Unlike OpenCL, Intel oneAPI architecture automatically implements optimizations in compilation time [Hagiescu and Cashman 2020], not only gaining much performance but also evidencing that our OpenCL time is misleadingly wrong. Another result, the C implementation for CPU performing better than OpenCL for GPU, might cause some confusion as we expected the opposite. We hypothesize that we did not use a sufficient large 2D workload to

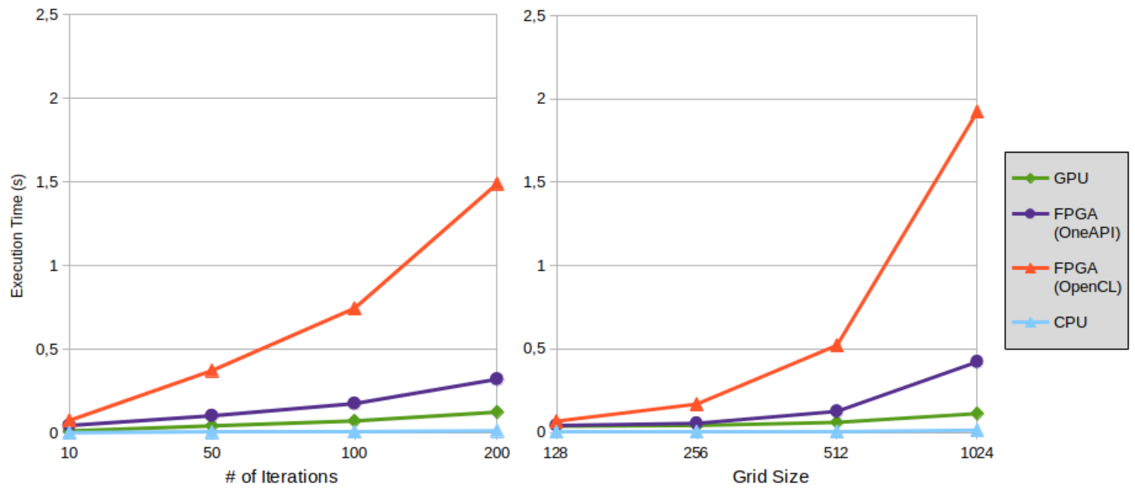

Figure 2. Execution Time in function of # of Iterations (left) and Grid Size (right).

pay the data transfer time to GPUs and FPGAs boards' memory. We would need a grid size at least a hundred times bigger or a 3D matrix for those devices to start getting ahead.

7. Conclusion

OpenCL and OneAPI make it easier to design an accelerator for stencil computation. However, although the algorithm is written in a C/C++ style language, the designer has to provide hardware architecture and optimization hints to the development tool to obtain a good computation performance. For this reason, we did not obtain good performance results for GPU and FPGA compared with execution in CPU. We expect to decrease the time gap between OpenCL and oneAPI. Moreover, we also expect to optimize both to bring it closer to GPU and CPU performance and to turn the FPGA a better option regarding energy efficiency, sustaining the idea of providing better options depending on the user objectives.

Acknowledgements

The authors gratefully acknowledge the sponsorship from Shell Brasil through the *ANP 20714-2 - Desenvolvimento de técnicas numéricas e software para problemas de inversão com aplicações em processamento sísmico* project at Universidade de São Paulo and the strategic importance of the support given by ANP through the R&D levy regulation.

References

Hagiescu, A. and Cashman, D. (2020). Accelerating compression on intel fpgas. *Tech Decoded*.

Justin Holewinski, L.-N. P. and Sadayappan, P. (2012). High-performance code generation for stencil computations on gpu architectures. ICS '12. Association for Computing Machinery.

Schäfer, A. and Fey, D. (2011). High performance stencil code algorithms for gpus. *Procedia Computer Science*. Proceedings of the International Conference on Computational Science, ICCS 2011.