International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Performance Evaluation and Enhancement of The Fletcher Method on Multicore Architectures

***Mathew Serpa*[1], *Priya Mishra*[2]**

[1,2]Departament of Computer Science - Siddaganga Institute of Technology, Tumakuru 572103, India

ABSTRACT

The simulation of acoustic wave propagation is the basis of seismic imaging tools used by the oil and gas industry. To perform such simulations, CAD architectures are employed, providing faster and more accurate results for each generation of processors. However, to achieve high performance in these architectures, several challenges must be taken into account when developing the application. In this article, Fletcher Modeling was optimized for multicore and GPU, and the performance, power consumption, and energy efficiency of eight versions of the code were evaluated. The results show that the CUDA version has the best performance and energy efficiency; however, the OpenACC version which has the advantage of portability, has a performance and energy efficiency degradation of only 10% and 8%compared to CUDA.

Keywords: GPU, Performance evaluation, Multicore, CUDA.

1. Introduction

Exploration geophysics has been fundamental in the search for energy resources, such as Gas and Oil. However, high drilling costs, with less than 50% accuracy per bit, limit its use [Qutob et al. 2004, Lukawski et al. 2014]. Thus, the Oil and Gas industry relies on software focused on High Performance Computing (CAD) as an economically viable way to reduce costs. The basis of many software engines for exploration geophysics is wave propagation simulation. For example, in seismic imaging, modeling, migration and inversion tools, this simulation can be used. These tools are built based on Partial Differential Equation (PDE) solvers, where the PDE solved in each case defines the accuracy of the approximation to real physics.

The approximation of the acoustic wave propagation model is the current basis of seismic imaging tools. It has been extensively applied in recent years in oil and gas reservoirs with imaging potential under saline domes. These applications must be continually ported to the newest CAD hardware available to remain competitive in the marketplace. At the same time, CAD architectures have evolved and performance improvement just by increasing clock frequency is no more. The solutions currently adopted are being replaced by multicore and manycore technologies [Clapp et al. 2010, Clapp 2015].

The last decade has seen a trend towards building high performance systems with dedicated devices and accelerators, which produce a good return on energy efficiency (FLOPs/Watt) [J. Dongarra and Strohmaier 2019]. Among the available alternatives, accelerator manufacturers have dedicated efforts to provide tens to thousands of processing units working at low frequencies, such as Graphics Processing Units (GPUs), and other accelerators [Witten et al. 2016]. Even more traditional multicore processors, such as the Intel Xeon family, include dozens of cores in the processors, working at high frequencies.

Several challenges must be taken into account in order to achieve high performance in these architectures. One of the most important aspects is the behavior of the memory subsystem, since memory access plays a key role in performance [Serpa et al. 2019b]. Energy consumption and energy efficiency are also points to consider [Subramaniam et al. 2013]. Still, heterogeneous computing has gained strength, increasing the complexity of using architectures. This increase in complexity occurs because these architectures have their own memories and cores that have different latencies. In addition, data must be copied by the programmer from one memory to another, increasing programming difficulty. From this, several more generic and high-level programming models emerged, which allow the writing of portable code, which can be executed on CPUGPU and GPU architectures without any changes [Sabne et al. 2014].

In this article, we work with Fletcher Modeling, a wave propagation application used by oil companies. Our goals are:

- Exploit the intrinsic parallelism of Fletcher Modeling to efficiently utilize multicore and GPU architectures;
- Evaluate the cost of portable versions from the point of view of performance, energy consumption and energy efficiency. The paper is organized as follows.

Section 2 discusses related work. Section 3 presents Fletcher Modeling and details of its implementation. Section 4 describes the architectures, metrics, and detailed information about the experiments. Section 5 provides an evaluation of the performance, power consumption and energy efficiency of the different versions of the application. Finally, Section 6 presents conclusions and future work.

2. Related Works

Recent architectures, including GPUsaccelerators, have proven to be suitable for geophysics, magnetohydrodynamics, and flow simulations, outperforming general-purpose processors in efficiency [Yuen et al. 2013]. To get maximum performance on these new devices, some reengineering of code regions, if not the entire application, is required. Thus, [Kukreja et al. 2016] automatically generates code for geophysical simulation that is highly optimized for multiple architectures, while [Niu et al. 2014] suggests the use of runtime system reconfiguration and a performance model to reduce the consumption of computational resources.

[Caballero et al. 2015] studied the effect of different memory and computational optimizations on elastic wave propagation equations. The results showed that comparing the processor and the Xeon Phi coprocessor, the processor was up to four times larger than the coprocessor.

[Rubio et al. 2013] rewrote a generic elastic wave propagator on general purpose processors, GPUs, and Xeon Phi, showing that the coprocessor provides good performance at a reduced development cost. Our optimizations target only one type of domain to reduce the complexity of the problem and to restrict the number of variables being reproduced in the analysis.

In [Andreolli et al. 2015], the authors focused on acoustic wave propagation equations, choosing the optimization techniques from the systematic adjustment of the algorithm. The use of thread blocking, cache blocking, register reuse, vectorization and redistribution of loop data resulted in significant performance improvements. Our proposal seeks to analyze and optimize a seismic image simulation widely used by oil companies such as Petrobras.

Research efforts like the one presented in [Castro et al. 2016] improved and evaluated the performance of the acoustic wave propagation equation on the Intel Xeon Phi coprocessor and compared it with the manycore processor MPPA-256, with general purpose processors and with a GPU. Optimizations include cache blocking, pointer-shifting memory alignment, and thread affinity. They show that the best results are obtained from a combination of the first two and that the performance with Xeon Phi is close to GPU. Our work goes a step further by studying the trade-off between portability, performance and energy efficiency of a real geophysics application.

[Pavan et al. 2018] analyzes and proposes optimizations for the input and output (I/O) operations of a geophysics application that uses the Reverse Time Migration algorithm (RTM). The results show that the use of checkpoints and increasing the size of the request reduces the application execution time by up to 17,33%. Once we identified that the I/O is not the main bottleneck of our application, we focused on the analysis of other factors related to performance, consumption and portability.

[Zhebel et al. 2013] compared the scalability of non-optimized code for finite differences and finite element algorithms on Intel Xeon and Xeon Phi. In Xeon, the scalability was similar and non-linear for all methods, while in Xeon Phi, only the finite difference presented lower scalability, due to the idleness of the I/O control thread. Our proposal goes beyond a scalability analysis and seeks a better understanding of the behavior of a real application in multicore and GPU architectures.

[Carrijo Nasciutti et al. 2018] performed a performance analysis of a synthetic matrix-based application 3Don GPUs focusing on the proper use of the memory hierarchy. They conclude that the highest performing code is what combines read-only cache reuse, inserting the dimension loop Zinto the kernel, and reusing processor registers. Unlike this work, we evaluated the performance and energy efficiency of a real application.

[Serpa et al. 2019a] proposes several optimization strategies for a wave propagation model for six CPUand GPU. The work focuses on improving cache memory usage, vectorization, load balancing and locality in the memory hierarchy. However, the work does not take into account energy consumption and energy efficiency.

3. Fletcher Modeling

The Fletcher Modeling application simulates the propagation of waves through time. Wave propagation is defined by the acoustic equation (Equation 1), and different geological layers have different velocities (Equation 2), where p(x,y,z,t)is the pressure at each point of the domain over time, V(x,y,z)is the propagation velocity and ρ(x,y,z)is the density. The deduction of partial differential equations, such as boundary conditions and stability, can be found in [Fletcher et al. 2009].

$$\frac{1}{V^2}\cdot\frac{\partial^2 p}{\partial t^2} = \nabla^2 p (1) \frac{1}{V^2}\cdot\frac{\partial^2 p}{\partial t^2} = \nabla^2 p - \frac{\nabla\rho}{\rho}\cdot\nabla p (2)$$

The modelling simulates data collection in a seismic survey, as illustrated in Figure 1. From time to time, equipment attached to a ship emits waves that reflect and refract in changes in the subsurface environment. Eventually, these waves return to the sea surface, being collected by specific microphones attached to cables towed by the ship. The set of signals received in each earphone over time constitutes a seismic trace. For each emission of waves, the seismic traces of all the handsets in the cable are recorded. The ship moves and emits signals over time.

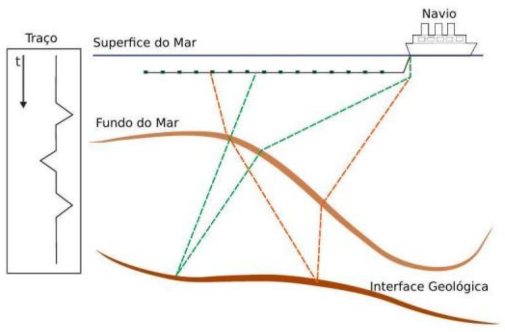


Figure 1. Data collection in a maritime seismic survey.

In the oil and gas industry, an isotropic wave propagation modeling program was written. Ocode was written in language Cand discretization was done using finite differences. Several versions of the code have been made as an OpenMP version [Chandra et al. 2001] to explore only multicore processors, a CUDA version [Sanders and Kandrot 2010] for GPUs and a single OpenA version CC[Wienke et al. 2012] for both architectures.

1.1. Original Version

There is a loop that represents the number of iterations of the application. In each iteration, first the wave sources are entered. Afterwards, the propagation is done through the calculation of a stencil, which is commonly used in this type of scientific applications to solve partial differential equations on multidimensional grids. These calculations are independent because each point is a combination of the values of a point and its neighbors in the previous iteration, as shown in Figure 2. For the computation of each of these points several other points need to be read from cache memories and main memory. This generates more memory accesses than computation per point. In this sense, memory access behavior presents challenges to optimize performance.

The code snippet advances the wave propagation at the internal points of a grid (cube). Three-dimensional fields are mapped to one-dimensional arrays in order (x,y,z), that is, it xis a direction that changes most rapidly when storing the arrays in memory. The direction of the derivative is determined by the memory jump (in the x, y and z directions). The parallelism of the original version is immediate, as the propagation at one point on the grid is independent of the propagation at any other point on the grid. Therefore, the nested loops that traverse the grid are fully parallelizable.

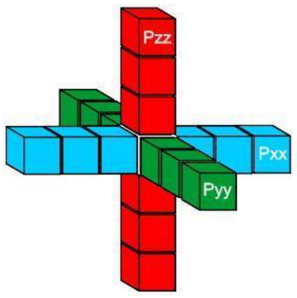


Figure 2. 7-point 3D stencil.

1.2. Optimized Version

It was identified that the calculation of cross derivatives required 76%floating point operations in the original version. This finding was obtained using the PAPI profiler tool [Terpstra et al. 2010] in sequential execution and in OpenMP execution of the original version of a 216×216×216point grid. A similar finding was obtained with the profiler tool NV Prof [Nvidia 2016] on the GPU on the same grid.

In this sense, we seek to reduce the number of operations to calculate the cross derivatives. The change made is to calculate the cross derivative in x y as the first derivative in y of the first derivative in x. This does not reduce the number of operations in computing the cross derivative at one point, but it does reduce the number of operations in computing the cross derivative on consecutive y s, by reusing seven previously calculated derivatives x out of the eight required. As the same reduction occurs in the other cross derivatives, this was applied to the calculation of all cross derivatives.

The first derivative at x was used to calculate the cross derivatives at x y and at x z, and the first derivative at y to calculate the cross derivative at y z. Since first derivatives are only used to calculate cross derivatives, values of these derivatives at a given depth z are only needed at some depths above and below. Therefore, the complete field of each first derivative that is stored in an array can be replaced by a circular buffer, which stores the values of this derivative in small depths, reducing the increase in memory consumption. Since the circular buffer is much smaller than the array that has the full field, it is possible to improve the use of the memory hierarchy, speeding up the computation. In this case, there is a parallel region containing the initialization, followed by the sequential loop at depths, containing two parallel regions, the first for computing the remaining derivative and the second for applying propagation.

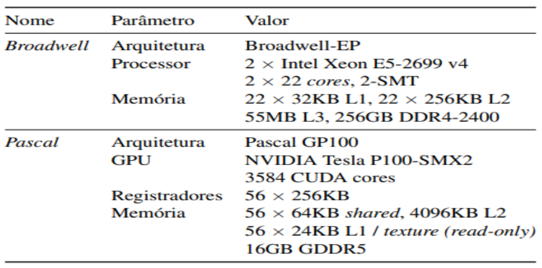
1.3. Assessment Methodology

The OpenMP, OpenACC and CUDA encodings were performed on 12 grids (cubes) of increasing sizes, chosen so that the size of the grid in any direction is a multiple of 32. The value 32 was chosen because it is the size of the CUDA warps. Since the results were similar to each other, we show in this work the results for a cube of 504×504×504, which is the largest size that fits in the main memory of the GPU. Each experiment was run 30 times with the number of virtual cores of each architecture. The graphs obtained show the mean values of execution time and the confidence intervals according to the 95%Student distribution [Ott and Longnecker 2015].

The experiments were carried out in Broadwell and Pascal environments. Broadwell has two 22-core Intel Xeon E5-2699 v4 processors. Each core supports 2-way Simultaneous Multithreading (SMT), allowing up to 88 threads to run. The cores have private L1 and L2 caches, while the L3 cache is shared between all cores of each processor. Pascal is an NVIDIA PIOO GPU with 3584 CUDA Cores. Table 1 has detailed information for each environment.

The results show performance in samples per second, energy consumption in Joules, and energy efficiency in Joules per sample. The performance is given by dividing the number of grid points calculated by the application execution time. The power consumption is calculated by the integral of the power consumed by the motherboard during the execution. This power is obtained via the Intelligent Platform Management Interface (IPMI)[Slaight 2002] which measures the power consumed by the motherboard and the GPU. Finally, energy efficiency is obtained by dividing the energy by the number of samples, that is, the amount of energy consumed to calculate one sample per second.

Table 1. Configuration of the evaluated architectures.



Nome	Parâmetro	Valor
Broadwell	Arquitetura	Broadwell-EP
	Processor	2 × Intel Xeon E5-2699 v4
		2 × 22 *cores*, 2-SMT
	Memória	22 × 32KB L1, 22 × 256KB L2
		55MB L3, 256GB DDR4-2400
Pascal	Arquitetura	Pascal GP100
	GPU	NVIDIA Tesla P100-SMX2
		3584 CUDA cores
	Registradores	56 × 256KB
	Memória	56 × 64KB *shared*, 4096KB L2
		56 × 24KB L1 / *texture (read-only)*
		16GB GDDR5

1.4. Experimental Results

Initially, we evaluated the performance of the two implementations of the application and, we discussed the compromise between the performance and portability of the different versions of the application over different parallel programming interfaces (IPPs). Afterwards, we evaluate energy consumption and energy efficiency.

1.5. Performance and Portability Assessment

A performance evaluation was performed to determine the cost of portability between multicore and GPU, contrasting the performance of portable encoding (OpenACC) in both architectures with that of non-portable encodings, specific to each architecture (OpenMP in multicore and CUDAin GPU). The portability of OpenACC is such that a single build generates code for both multicore and GPU. An environment variable defines, during execution, the architecture to use.

Figure 3 shows the performance in millions of samples per second, for OpenMP and OpenA encodings CCrunning on CPU,CUDAand OpenACC running on GPU. In the original version, OpenACC is 12,9%faster than OpenMP and 57.5%slower than CUDA. In the optimized version, OpenACC is 36%faster than OpenMP and 10.5%slower than CUDA. This shows that our portable version (OpenACC multicorelGPU) is faster than OpenMP on multicore, but does GPUn't lose performance compared to CUDA.

We also analyzed the gain of the optimized version over the original. Operformance of the optimized version using OpenMP was 48.3% better than the original OpenMP version. In the case of version CUDA, the gain was 73.6%. The OpenACC version benefited the most from this optimization. The performance improvement was 78,7%in the OpenA CC-CPU 265,9%version and in the OpenACC-GPU version, approaching the performance of the CUDA. Furthermore, the results show that replacing the cross-derivative with the first derivative of the first derivative is more fruitful for GPUthan for multicore.

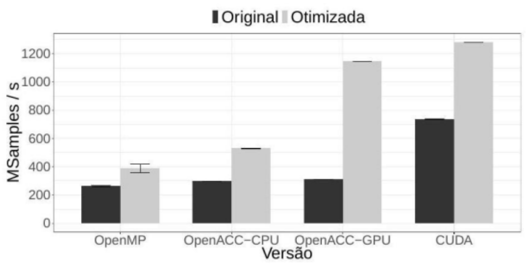


Figure 3. Performance of the original and optimized versions in different IPPs.

1.6. Energy Consumption and Energy Efficiency

Figure 4 shows the average power consumption of the entire machine in Kilo joules for OpenMP and OpenA encodings running on CPU, CUDA and OpenACCrunning on GPU. We can notice that, in general, the optimized version presents a reduction in the average energy consumption in all implementations, on average the original version consumed 12,5" " and the optimized version 10,8" " kJ, this represents a reduction of 10,8%in energy consumption.

Another aspect that we can notice is that the parallel programming interfaces have different behaviors from each other. Analyzing the implementation in OpenMP for multicore, the optimized version achieved a reduction in 6,0%energy used. While the OpenACC implementation had a reduction of 24,5% in multicore and 21,4%GPU. The implementation CUDAwas the one that had the smallest reduction in consumption, about 3,6%.

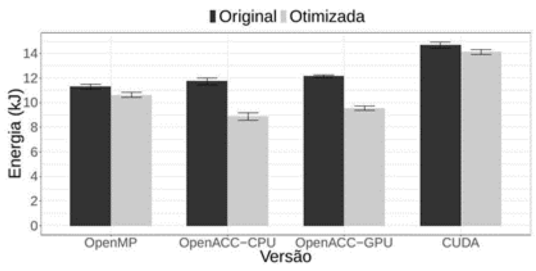


Figure 4. Power consumption of the original and optimized versions.

Seeking to analyze which implementation has the best energy efficiency, Figure 5 presents the nano joule (nJ)per sample ratio for each implementation. Observing the general relationship between the original and the optimized version, the original presented on average, 1066" " nJ/sample, while the optimized presented on average 537" " nJ/sample, this represents an increase in efficiency of 49,6%.

Analyzing the energy efficiency between the parallel programming interfaces and the original and optimized versions, it is possible to verify that OpenMP has a better efficiency of 28,9%in the optimized version compared to the original. In the case of the Open version ACC, energy efficiency has been improved in 52,8% multicore GPU and 74,6%. The optimized CUDA implementation had better efficiency 42,6%than the original version.

Comparing energy consumption with energy efficiency, we can see that despite the OpenMP and OpenACC encodings consuming less energy than the CUDA version, they presented worse energy efficiency, in relation to it, in the original version, where the version CUDA was even 67,5%more efficient than the CUDA version. the other encodings.

CUDA encoding showed the best efficiency in the optimized version, followed by Open encoding ACC-GPU with a difference of 8,9%. Writing code in OpenACC is simpler than in CUDA[Memeti et al. 2017] and the encoding is portable, that is, the same encoding can be used in both multicore and GPU. However, we show that there are losses both in performance and in energy consumption and energy efficiency.

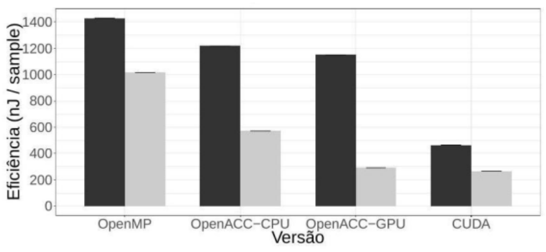


Figure 5. Energy Efficiency of the original and optimized versions.

4. Conclusions

Current multicore and GPU architectures introduce several challenges, such as the need to properly code parallel applications to get the best out of them. In this article, we optimize a wave propagation application, Fletcher Modeling, for multicore Intel Xeon and NVIDIA Pascal GPU architectures. We showed that the calculation of cross derivatives required 76%floating point operations and because of this, we propose an optimization to reduce this cost. This optimization can also be applied in other similar applications and other architectures.

In our experiments, we also analyzed the performance, power consumption, and energy efficiency of the application. In addition to doing a study on the feasibility of using a portable version of the code. The results showed that the version with the highest performance was the CUDA version, followed by the OpenACC-GPU version. The performance difference between the CUDA version and the OpenACC-GPU version, which can run on different architectures, was 10%. This shows that there is a cost in using a portable version, but depending on the test case, it can be interesting. Afterwards, we show that energy consumption is not the only determining metric, since an application can consume more energy, running quickly and thus, having the best energy efficiency. In this sense, we analyzed the energy efficiency, and showed that the CUDA and OpenACC versions had the best efficiency, 266 and 292 nanojoules per sample, respectively.

As future work, we intend to evaluate architectures with FPGAintegrated boards programmed with OpenCL. We are also interested in verifying the efficiency of using multiple GPUs on the same node and comparing our results with the solutions proposed in the literature.

References

1. Andreolli, C., Thierry, P., Borges, L., Skinner, G., and Yount, C. (2015). Characterization and Optimization Methodology Applied to Stencil Computations. In Reinders, J. and Jeffers, J., editors, High Performance Parallelism Pearls, pages 377-396. Morgan Kaufmann, Boston.
2. Caballero, D., Farrés, A., Duran, A., Hanzich, M., Fernández, S., and Martorell, X. (2015). Optimizing Fully Anisotropic Elastic Propagation on Intel Xeon Phi Coprocessors. In 2nd EAGE Workshop on HPC for Upstream, pages 1-6.
3. Carrijo Nasciutti, T., Panetta, J., and Pais Lopes, P. (2018). Evaluating optimizations that reduce global memory accesses of stencil computations in gpgpus. Concurrency and Computation: Practice and Experience, page e4929.
4. Castro, M., Francesquini, E., Dupros, F., Aochi, H., Navaux, P. O. A., and Méhaut, J.-F. (2016). Seismic wave propagation simulations on low-power and performance-centric manycores. Parallel Computing, 54.
5. Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., and McDonald, J. (2001). Parallel programming in OpenMP. Morgan Kaufmann.
6. Clapp, R. G. (2015). Seismic Processing and the Computer Revolution(s). In Society of Exploration Geophysicists (SEG) Technical Program Expanded Abstracts 2015, pages 4832-4837.
7. Clapp, R. G., Fu, H., and Lindtjorn, O. (2010). Selecting the right hardware for reverse time migration. The Leading Edge, 29(1).
8. Fletcher, R. P., Du, X., and Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic media. Geophysics, 74(6):WCA179–WCA 187.
9. J. Dongarra, H. M. and Strohmaier, E. (2019). Top500 supercomputer: June 2019. hetps://Www . top 500 . org/lists / 2019/06/. [Acesso em: 10 Jul. 2019].
10. Kukreja, N., Louboutin, M., Vieira, F., Luporini, F., Lange, M., and Gorman, G. (2016). Devito: Automated fast finite difference computation. In Procs. of the 6th Intl. Workshop on Domain-Spec. Lang. and High-Level Frameworks for HPC, WOLFHPC '16, pages 11-19. IEEE Press.
11. Lukawski, M. Z., Anderson, B. J., Augustine, C., Capuano Jr, L. E., Beckers, K. F., Livesay, B., and Tester, J. W. (2014). Cost analysis of oil, gas, and geothermal well drilling. Journal of Petroleum Science and Engineering, 118:1-14.
12. Memeti, S., Li, L., Pllana, S., Kołodziej, J., and Kessler, C. (2017). Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption. In Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, pages 1-6. ACM.
13. Niu, X., Jin, Q., Luk, W., and Weston, S. (2014). A Self-Aware Tuning and SelfAware Evaluation Method for Finite-Difference Applications in Reconfigurable Systems. ACM Trans. on Reconf. Technology and Systems, 7(2).
14. Nvidia (2016). Developer Zone - CUDA Toolkit Documentation.
15. Ott, R. L. and Longnecker, M. T. (2015). An introduction to statistical methods and data analysis. Nelson Education.
16. Pavan, P. J., Serpa, M. S., Padoin, E. L., Schnorr, L. M., Navaux, P. O. A., and Panetta, J. (2018). Improving i/o performance of rtm algorithm for oil and gas simulation. In 2018 Symposium on High Performance Computing Systems (WSCAD), pages 270-270. IEEE.
17. Qutob, H. et al. (2004). Underbalanced drilling; remedy for formation damage, lost circulation, & other related conventional drilling problems. In Abu Dhabi International Conference and Exhibition. Society of Petroleum Engineers.
18. S. Dutta, S. Manakkadu, and D. Kagaris. "Classifying performance bottlenecks in multi-threaded applications." In 2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, pp. 341-345. IEEE, 2014.
19. S. Manakkadu, and S. Dutta. "Bandwidth based performance optimization of Multi-threaded applications." In 2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming, pp. 118-122. IEEE, 2014.
20. Kavi, Krishna M., Roberto Giorgi, and Joseph Arul. "Scheduled dataflow: Execution paradigm, architecture, and performance evaluation." IEEE Transactions on Computers 50, no. 8 (2001): 834-846.
21. Islam, Mahzabeen, Marko Scrbak, Krishna M. Kavi, Mike Ignatowski, and Nuwan Jayasena. "Improving node-level mapreduce performance using processing-in-memory technologies." In European Conference on Parallel Processing, pp. 425-437. Springer, Cham, 2014.
22. T. Janjus, K. Krishna, and B. Potter. "International conference on computational science, iccs 2011 gleipnir: A memory analysis tool." Procedia Computer Science 4 (2011): 2058-2067.

23. Rubio, F., Farrés, A., Hanzich, M., de la Puente, J., and Ferrer, M. (2013). Optimizing Isotropic and Fully-anisotropic Elastic Modelling on Multi-GPU Platforms. In 75 th EAGE Conference & Exhibition, pages 10-13. EAGE.
24. Sabne, A., Sakdhnagool, P., Lee, S., and Vetter, J. S. (2014). Evaluating performance portability of openacc. In International Workshop on Languages and Compilers for Parallel Computing, pages 51-66. Springer.
25. Sanders, J. and Kandrot, E. (2010). CUDA by example: an introduction to generalpurpose GPU programming. Addison-Wesley Professional.
26. Serpa, M. S., Cruz, E. H., Diener, M., Krause, A. M., Navaux, P. O. A., Panetta, J., Farrés, A., Rosas, C., and Hanzich, M. (2019a). Optimization strategies for geophysics models on manycore systems. The International Journal of High Performance Computing Applications, 33(3):473-486.
27. Serpa, M. S., Moreira, F. B., Navaux, P. O., Cruz, E. H., Diener, M., Griebler, D., and Fernandes, L. G. (2019b). Memory performance and bottlenecks in multicore and gpu architectures. In 201927 th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pages 233-236. IEEE.
28. Slaight, T. (2002). Platform management ipmi controllers, sensors, and tools. In Intel Developer Forum.
29. Subramaniam, B., Saunders, W., Scogland, T., and Feng, W.-c. (2013). Trends in energyefficient computing: A perspective from the green500. In 2013 International Green Computing Conference Proceedings, pages 1-8. IEEE.
30. Terpstra, D., Jagode, H., You, H., and Dongarra, J. (2010). Collecting performance data with papi-c. In Tools for High Performance Computing 2009, pages 157-173. Springer.
31. Wienke, S., Springer, P., Terboven, C., and an Mey, D. (2012). Openacc-first experiences with real-world applications. In European Conference on Parallel Processing, pages 859-870. Springer.
32. Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.
33. Yuen, D. A., Wang, L., Chi, X., Johnsson, L., Ge, W., and Shi, Y. (2013). GPU solutions to multi-scale problems in science and engineering. Springer.
34. Zhebel, E., Minisini, S., Kononov, A., and Mulder, W. (2013). Performance and scalability of finite-difference and finite-element wave-propagation modeling on Intel's Xeon Phi. In Society of Exploration Geophysicists (SEG) Technical Program Expanded Abstracts 2013, pages 3386-3390.