Improving Performance and Energy Efficiency of Geophysics Applications on GPU Architectures

Pablo J. Pavan[1], Matheus S. Serpa[1], Emmanuell D. Carreño[2], Víctor Martínez[1], Edson L. Padoin[1,3], Philippe O. A. Navaux[1], Jairo Panetta[4], and Jean-François Mehaut[5]

[1] Informatics Institute
Federal University of Rio Grande do Sul – UFRGS
Porto Alegre, Brazil
{pjpavan, msserpa, victor.martinez, navaux}@inf.ufrgs.br

[2] Department of Informatics
Federal University of Paraná – UFPR
Paraná, Brazil
edcarreno@inf.ufpr.br

[3] Department of Exact Sciences and Engineering
Regional University of the Northwest of the State of Rio Grande do Sul – UNIJUI
Ijuí, Brazil
padoin@unijui.edu.br

[4] Computer Science Division
Technological Institute of Aeronautics – ITA
São José dos Campos, Brazil
jairo.panetta@gmail.com

[5] Laboratoire d'Informatique de Grenoble
University of Grenoble – UGA
Grenoble, France
jean-francois.mehaut@imag.fr

Abstract. Energy and performance of parallel systems are an increasing concern for new large-scale systems. Research has been developed in response to this challenge aiming the manufacture of more energy efficient systems. In this context, this paper proposes optimization methods to accelerate performance and increase energy efficiency of geophysics applications used in conjunction to algorithm and GPU memory characteristics. The optimizations we developed applied to Graphics Processing Units (GPU) algorithms for stencil applications achieve a performance improvement of up to 44.65% compared with the read-only version. The computational results have shown that the combination of use read-only memory, the Z-axis *internalization* and reuse of specific architecture registers allow increase the energy efficiency of up to 54.11% when shared memory was used and increase of up to 44.53% when read-only was used.

Keywords: Geophysics applications · Manycore systems · Energy efficiency · GPU.

1 Introduction

Several applications in areas, such as physics simulation, weather forecast, oil exploration, climate modeling and atomic simulation require high processing power and efficient models. Some of these scientific applications make use of stencil computations that include both implicit and explicit Partial Differential Equations (PDE) solvers [3]. Besides the scientific importance of stencils, they are interesting as an architectural evaluation benchmark because they have abundant parallelism and low computational intensity, offering opportunities for on-chip parallelism and challenges for associated memory systems [3]. Today, PetaFlops systems allow reaching increasingly accurate results these scientific applications. To respond to the high processing demand of stencil applications, High Performance Computing (HPC) systems gather the processing power of several computational resources to solve these problems.

Scientific simulations may consume weeks of supercomputer time and most of this time is spent in stencil computations [2]. Continuous changes in the fabrication process of the microprocessors industry have increased the performance of its products and influenced state-of-the-art HPC systems. However, this exponential increase in computational performance also leads to an exponential growth in power demand [4, 8, 13]. Reductions in the total execution time of applications are also relevant for energy consumption, energy is saved when hardware resources are used for a shorter time.

However, it is possible to achieve even greater energy savings if the application is able to exploit the different memory levels available. Today, the combined use of Graphics Processing Units (GPUs) and CPUs in HPC systems has become a popular choice among the top ranked and yet to come platforms. Stencil computing is typically memory-bound, memory performance is particularly important for most stencil kernels. GPUs have several processing elements inside a single die and different memory levels. For this reason, one of the most important strategies for optimizing the performance of stencil computing is the optimization of memory access. Besides, stencil computing can be ported to GPUs with significantly improved performance when compared to implementations performed on CPUs [9].

The performance of a stencil for a given architecture can be estimated through the roofline model [16]. This model relates the maximum performance of code to its computational intensity, considering the speed of memory access and the processing capacity of the machine [11]. In this paper, we improved the performance and achieved increase energy efficiency of stencil applications by improving methods and optimizations of GPU code. Those are used in conjunction with specific GPU memory characteristics. We focus on analyzing the impact of the stencil size and usage of different memory hierarchies and registers of the GPU to improve performance, power demand, energy consumption and energy efficiency.

The remaining sections of this paper are organized as follows. Section 2 discusses some of the related works on energy consumption In Section 3, we present the stencils application and details of our versions and optimization developed.

In Section 4, we present the evaluation methodology used in the conducted experiments and the stencils and their implementation details. In addition, in Section 5, we address the results obtained from the experiments. Finally, the Section 6 emphasizes the scientific contribution of the work and notes several challenges that we can address in the future.

2 Related Work

Several studies have evaluated performance of stencils to improve their energy efficiency in CPUs and GPUs systems. Despite that, the processors and accelerators remain as the component with the highest power demand of the systems [6]. GPUs are made aiming at massively parallel processing, to achieve this they use hundreds of processing units working together. These characteristics lead to its superior energy efficiency if compared with CPUs systems [14].

Micikevicius *et al.* [10] compared the performance of a stencil ported from CPU to GPU. Their version of the stencil running in a GPU achieved an order of magnitude higher than running in a contemporary CPU. They conclude that it is possible to improve their results by the usage of shared memory to reduce communication overhead.

Bauer *et al.* [1] showed that the main bottleneck in GPU applications are related to the memory system. To reduce its impact, they used DMA warps to improve memory transfer between on-chip and off-chip memories. They achieved a speedup up to 3.2 times on several kernels form scientific applications.

Schäfer and Fey [15] evaluate a set of algorithms on Fermi GPUs. They evaluate micro benchmarks using shared memory and found that using only L1 cache creates a problem for its limited throughput. Also, the L2 cache is not a good option because of cache blocking. They conclude that a new alternative to use shared memory was needed to overcome communication bottleneck.

Falch and Elster [5] proposed the usage of a manually managed cache to combine the memory from multiple threads. Using their technique, they achieved a speedup of up to 2.04 in a synthetic stencil. They concluded that manual caching is an effective approach to improve memory access and that applications with regular access patterns are suitable to implement their technique.

Zhou *et al.* [18] points that the use of GPUs enables considerable gains in performance compared to using CPU. They have applied GPUs successfully in many computations and memory intensive realms due to its superior performances in float-pointing calculation, memory bandwidth, and power consumption. The results obtained show a speedup of up to 50 times using GPU algorithm rather than CPU algorithm. In similar works, Zhou *et al.* [19] obtained a speedup between 10 and 15 times using a GPU rather than CPU.

Xue *et al.* [17] also make comparisons between GPU and CPU implementation. They obtained a speedup up to 18 times in the GPU-based implementation of a time-reversal imaging micro-seismic event location.

Also, Nikitin *et al.* [12] obtained average speedup up to 46 times using GPU for compared to CPU for processing a synthetic seismic data set (data compression, de-noising, and interpolation).

Maruyama and Aoky [9] presents a method for stencil computations on the NVIDIA Kepler architecture that uses shared memory for better data locality combined with warp specialization for higher instruction throughput, their method achieves approximately 80% of the value from roof line model estimation.

Hamilton *et al.* [7] investigate the computational performance of GPU-based stencil operations using stencils of varying shape and size (ranging from seven to more than 450 points in size). They found that using an NVIDIA K20 GPU, data movement, rather than computing, was the bottleneck, and as such, the performance obtained can be attributed to the effects of the L2 and texture caches on the card.

Compact stencils are more efficient using the texture cache and require fewer reads from global memory. The leggy stencils schemes required a significant portion of global memory bandwidth in order to achieve similar performance as compact stencils of similar size in points.

Nasciutti and Panetta [11] did a performance analysis of 3D stencils on GPUs focusing on the proper use of the memory hierarchy. They conclude that the preferred code is the combination of read only cache reuse, inserting the Z loop into the kernel and register reuse.

Different to other approaches that allocate workload on CPU and GPU architectures, or works that use GPUs to achieve considerable performance gains when compared to traditional CPU architecture, our goal aims to increase the performance and energy efficiency of stencil application applying methods and optimization to use different memory levels of the GPUs.

3 Geophysical Model Optmizations

The model simulates the collection of data in a seismic wave propagation. At intervals of, equipment coupled to the ship emits waves that reflect and refract on changes of the medium in the subsoil. Eventually, these waves return to the surface of the sea, being collected by specific microphones (geophones) coupled to cables towed by the ship. The set of signals received by each geophone over time constitutes a seismic trace. For each wave emission, the seismic traces of all cable geophones are recorded. The ship continues to sailing and emits signals over time.

Acoustic wave propagation approximation is the current backbone for seismic imaging tools. It has been extensively applied to imaging potential oil and gas reservoirs beneath salt domes. We consider the model formulated by the isotropic acoustic wave propagation under Dirichlet boundary conditions over a finite 3D rectangular domain, prescribing to all boundaries, and the isotropic acoustic wave propagation. Propagation speed depends on variable density, the acoustic pressure, and the media density. These applications are modeling and solved using stencil computations.

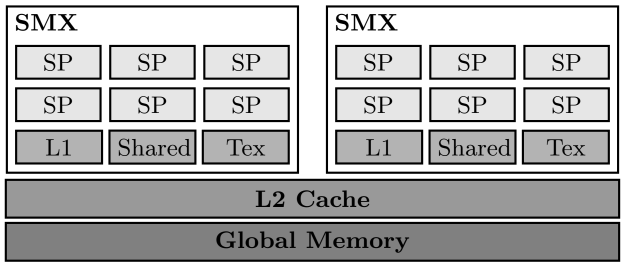


Fig. 1: Sample of the memory subsystem on a NVIDIA Kepler architecture

In this context, the computational performance of GPU-based stencils have a great scientific importance as it is used in many areas of scientific computing. Regarding the capatiblities of current GPU architectures, the NVIDIA Kepler provide memories with different characteristics compared with CPUs. One of the main differences between GPUs and CPUs is the way their memory subsystem work. In a CPU, access to memory is done by obtaining their data from caches. Usually looking on L1, L2, L3, and DRAM in that order. On the other hand, in a GPU the L1 memory cache, is used specifically for accesses to the stack and register spill, i.e., when too many local variables do not fit in the register file, and thus some of it has to be cached. L2 memory is used for global accesses requested by stream processors.

The current GPU have also registers files, a shared memory. They are a texture memory and a global memory with different characteristics such as size, speed, read-only memory and in the way that is possible to use them. These registers were not available in Nvidia GPUs before Kepler architecture. In Figure 1 is shown an overview of the Kepler GPU architecture, which have different SP (Stream Processor) in each SMX (Streaming Multiprocessors).

To exploit the use of different memory levels available on current GPU, we develop three versions of a stencil kernel using each one of the GPU memories. Each stencil version, give us a different insight of the performance and capabilities of the GPU memory subsystem.

- The first version called *naive* take no advantage of any of the GPU high-speed memories and access data only from global memory.
- The second version called *shared* stores one part of the stencil data in the shared memory scratchpad. The shared-memory version also uses the GPU resources that the naive version uses, the main difference is that this version also uses the shared-memory available on each SMX (Streaming Multiprocessors). Each one of the SMX have one internal shared-memory to store data as shown in Figure 1. In this version, data is manually allocated by the programmer through the use of the *shared* directive, indicating such data will be shared among all the GPU threads. The compiler automatically configures the space division between the L1 cache memory and the shared

cache memory, choosing one of three options: 16 KB for the L1 cache and 48 KB for the shared cache, 32 KB for each, or 48 KB for the L1 cache and 16 KB for the shared cache.

– The third version called *read-only* stores most read data in a read-only texture memory which is faster than shared memory but works with read-only data. This version takes advantage of the read-only cache, this cache is the SMX memory bank that stores only read data, it is also called texture memory. Originally it was used only for textures, but starting with the Kepler architecture any data can be stored in this cache by using the C-99 directive `const restrict`. The programmer may also explicitly use this cache through the intrinsic `lgd()`.

We developed two optimizations for each of the versions to evaluate improvements in performance and energy efficiency by reusing the Z direction data. Reusing Z direction data is named *internalization*.

– The *int.z* version takes advantage of data locality by storing stencil data for direction Z. This optimization consists of the internalization of the Z-axis into the threads. Doing the internalization ensures that neighbouring Z-blocks execute sequentially, increasing the reuse of L2 cache data. Direction Z data is used to calculate subsequents points in the X-Y direction.
– The *int.z.reg* version consists of combining the *int.z* with the usage of registers to store the Z direction points. For example, to calculate the point Z3 in a 13 points stencil, the neighbouring points in X and Y, as well as points Z1, Z2, Z3, Z4 and Z5 are required. In order to calculate the points in Z4, points Z2, Z3, Z4 and Z5 would be availed, and it is necessary to request the global memory only points Z6, as well as the neighbours in X and Y.

4 Experimental Methodology

Our experiments were developed in a NVIDIA K20m GPU card. The K20m is a Kepler architecture GPU with 2496 CUDA cores. Each Streaming Multiprocessor has a configurable on-chip memory that can be configured as 48/32/16 KB shared memory with 16/32/48 KB of L1 cache. They also have a faster 48 KB read-only cache and a 1280 KB shared L2 cache. Table 1 describes in detail the environment we used.

We used NVIDIA Management Library (NVML) to measure the power usage. Regarding the energy efficiency measurement, we used the metric of performance achieved divided by average power. Each experiment was executed 30 times, we show average values, as well a 95% confidence interval calculated with Student's t-distribution.

5 Results

This section shows the optimizations techniques we used to improve the performance and energy efficiency of a stencil application. The stencil we used

Parameter	Value
Device	Tesla K20m
CUDA Cores	2496 (13 SMXs $\times$ 192 SPs/SMX)
Registers	13 $\times$ 256 KByte
Cache L1	13 $\times$ 64 KByte
Cache L2	*shared*, 1280 KByte
Texture (read-only)	13 $\times$ 48 KByte
Global Memory	5 GByte GDDR5

Table 1: Configuration of GPU system.

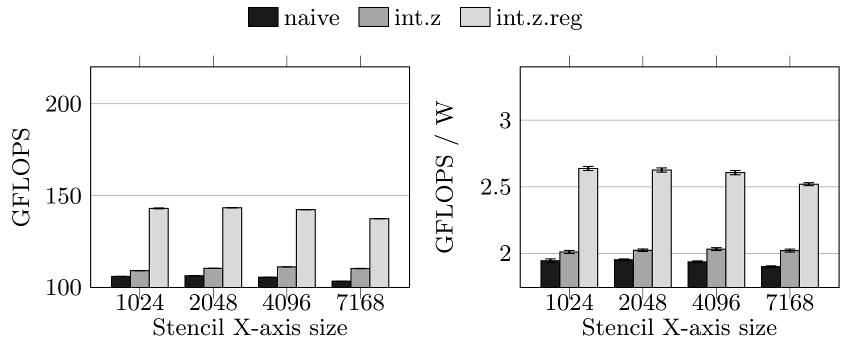
simulates the propagation of a single wavelet over time. To create the simulation, it solves the isotropic acoustic wave propagation with constant density under Dirichlet boundary conditions over a 3D domain. The stencil is a 13-arm with the following input sizes: $(1024 \times 256 \times 256)$, $(2048 \times 256 \times 256)$, $(4096 \times 256 \times 256)$, and $(7168 \times 256 \times 256)$.

In the following subsections, we describe each optimization and analyze how they address the performance and energy efficiency improvements. We also show the results obtained by using the three different memories and the results of the optimizations applied in each of them, on a NVIDIA Kepler architecture.

5.1 Performance and Energy Efficiency improvements over Naive version

This subsection shows the improvements obtained by using two optimization techniques over the naive version of the stencil computation. Figure 2 shows the performance and energy efficiency of the naive version and the optimizations. The first optimization technique used was the *int.z* which stores data from direction Z in local variables aiming to take advantage of the data locality by reusing these data in the subsequent iterations. The performance and energy efficient were improved by up to 4.65% and up to 4.55% using the *int.z* technique over a naive version. This improvement occurs due to the reuse of L2 cache data made by this optimization. The number of access in global memory is reduced by increasing L2 cache hits.

We propose a second optimization called *int.z.reg* which consists of the *int.z* optimization along with the use of the register file to store the Z points. In *int.z* Z points were stored only in local variables. Using this optimization performance overtakes the previous versions with an improvement of up to 34.31% compared with the naive version. The energy efficiency was also improved by up to 34.30%. The results show that the usage of registers, which are faster than local variables, allow us to obtain more performance with a better energy efficency.



(a) Performance gain. (b) Energy Efficiency gain.

Fig. 2: Improvements over Naive version which uses Global Memory.

5.2 Performance and Energy Efficiency improvements over Shared Memory

In the previous subsection, we showed the optimizations applied in the naive version. Although the performance and energy efficiency was improved by both optimizations techniques, the naive version does not take advantage of fast GPU memories as shared memory. Thus, we improved the naive version by using the shared memory scratch pad to store a slice of data that is reused by the threads of the same block. The data was manually allocated using the *shared* directive, indicating a piece of data shared among all threads. We also applied the *int.z* and *int.z.reg* optimizations aiming to improve the performance of the memory operations.

The performance and energy efficiency results are showed in Figure 3. Using this optimization, performance was improved by up to 2.25% and 54.46% in the *int.z* and *int.z.reg* optimizations compared with the shared memory version. It occurs due to the data stored in scratch pad is reused by the threads in the following iterations. The energy efficiency was improved by up to 2.02% and 54.11% using these optimizations.

5.3 Performance and Energy Efficiency improvements over Read-only Memory

In this subsection, we are showing the improvements obtained when we use both optimizations and the read-only memory. Since the data we store in the shared memory was not update we may take advantage of the read-only memory. The read-only memory is faster than shared memory but exclusively used for read-only operations. We can explicitly define that global memory reads be stored in the read-only memory using the *lgd()* intrinsic.

The *int.z* optimization over the *read.only* version achieve a performance improvement of up to 34.30%. Implementing the *int.z.reg* that also uses the register

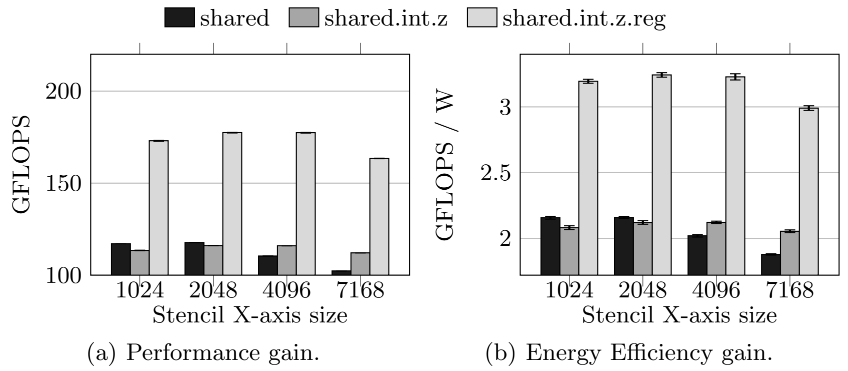


Fig. 3: Improvements over Shared Memory.

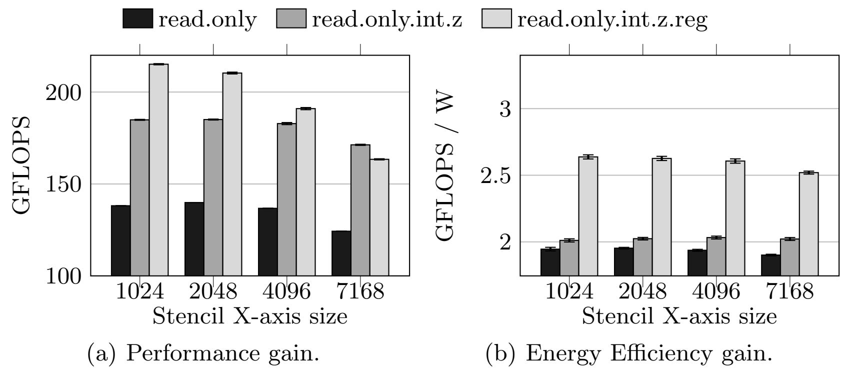


Fig. 4: Improvements over Read-only Memory.

file the performance was improved by up to 44.65%. The energy efficiency was also improved by these optimizations. The *int.z* version improved the energy efficiency by up to 34.20% while the *int.z.reg* improved the efficiency by up to 44.53%.

6 Conclusion

Several scientific applications make use of stencil computations to their model simulations. Stencils have both implicit and explicit PDE being so also interesting as an architectural evaluation benchmark. The computing present in these applications are low intensity, once that they are typically memory-bound. In this form, memory optimizations are important for to use the fastest memories available in GPUs and increase their the energy efficiency.

In this paper, aim to achieve energy savings, we introduce methods and optimization to stencil application that exploit the different memory levels available. Our developed methods, which are used in conjunction with specific GPU memory characteristics, allow to use the *read-only cache* and also the *shared memory*. Also, our developed optimization allows to combine the *Z-axis internalization* of stencil application with the *reuse of registers* of GPU architecture.

The main contribution of this paper is performance and energy efficiency increases when applied GPU-algorithms and optimization over stencil application. Our developed GPU-optimized algorithms for stencil applications achieve performance improvement of up to 54.11% and 44.53% when were used *shared memory* and *read-only cache* respectively over the naive version. This increase in computational performance also improves the energy efficiency in an equivalent value, once that our methods and optimization do not increase the power demand.

Changes in the GPU architecture, as in the case of the introduction of the read-only cache in the Kepler architecture, can generate changes in the results presented in this work. In the future, we plan to investigate methods and optimization to achieve gains in stencil applications over new NVIDIA architecture and Intel Xeon Phi.

Acknowledgments

This research has received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the *HPC4E Project*, grant agreement n.º 689772. It was also supported by Intel under the Modern Code project, and the *PETROBRAS* oil company under Ref. 2016/00133-9. We also thank to *RICAP*, partially funded by the Ibero-American Program of Science and Technology for Development (*CYTED*), Ref. 517RT0529.

References

1. Bauer, M., Cook, H., Khailany, B.: Cudadma: Optimizing gpu memory bandwidth via warp specialization. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 12:1–12:11. SC '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2063384.2063400, http://doi.acm.org/10.1145/2063384.2063400
2. de la Cruz, R., Araya-Polo, M.: Towards a multi-level cache performance model for 3d stencil computation. Procedia Computer Science **4**, 2146–2155 (2011)
3. Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., Yelick, K.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. p. 4. IEEE Press (2008)
4. Dong, Y., Chen, J., Tang, T.: Power measurements and analyses of massive object storage system. In: Proceedings of CIT. pp. 1317–1322. International Conference on Computer and Information Technology (CIT), IEEE Computer Society (2010). https://doi.org/10.1109/CIT.2010.237

5. Falch, T.L., Elster, A.C.: Register caching for stencil computations on gpus. In: 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 479–486. IEEE (Sept 2014). https://doi.org/10.1109/SYNASC.2014.70
6. Feng, X., Ge, R., Cameron, K.W.: Power and energy profiling of scientific applications on distributed systems. In: International Parallel and Distributed Processing Symposium (IPDPS). pp. 34–34. International Conference on Performance Engineering, IEEE (2005). https://doi.org/10.1109/IPDPS.2005.346
7. Hamilton, B., Webb, C.J., Gray, A., Bilbao, S.: Large stencil operations for gpu-based 3-d acoustics simulations. Proc. Digital Audio Effects (DAFx),(Trondheim, Norway) (2015)
8. Laros, J., Pedretti, K., Kelly, S., VanDyke, J., Ferreira, K., Vaughan, C., Swan, M.: Topics on measuring real power usage on high performance computing platforms. In: Proceedings... pp. 1–8. International Conference on Cluster Computing and Workshops (ICCC) (2009). https://doi.org/10.1109/CLUSTR.2009.5289179
9. Maruyama, N., Aoki, T.: Optimizing stencil computations for nvidia kepler gpus. In: Proceedings of the 1st International Workshop on High-Performance Stencil Computations, Vienna. pp. 89–95 (2014)
10. Micikevicius, P.: 3d finite difference computation on gpus using cuda. In: Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units. pp. 79–84. GPGPU-2, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1513895.1513905, http://doi.acm.org/10.1145/1513895.1513905
11. Nasciutti, T.C., Panetta, J.: Impacto da arquitetura de memria de gpgpus na velocidade de computao de estnceis. In: XVII Simpsio de Sistemas Computacionais (WSCAD-SSC). pp. 1–8. Aracaju, SE (2016)
12. Nikitin, V.V., Duchkov, A.A., Andersson, F.: Parallel algorithm of 3d wave-packet decomposition of seismic data: Implementation and optimization for gpu. Journal of Computational Science **3**(6), 469–473 (2012)
13. Padoin, E.L., de Oliveira, D.A.G., Velho, P., Navaux, P.O.A., Mehaut, J.F.: ARM-based cluster: Performance, Scalability and Energy Efficiency. In: 4th Workshop on Applications for Multi-Core Architectures (WAMCA SBAC-PAD). pp. 1–6. Porto de Galinhas, PB, Brasil (2013)
14. Padoin, E.L., Pilla, L.L., Boito, F.Z., Kassick, R.V., Velho, P., Navaux, P.O.: Evaluating application performance and energy consumption on hybrid cpu+ gpu architecture. Cluster Computing **16**(3), 511–525 (2013)
15. Schafer, A., Fey, D.: High performance stencil code algorithms for gpgpus. Procedia Computer Science **4**, 2027 – 2036 (2011). https://doi.org/http://dx.doi.org/10.1016/j.procs.2011.04.221, http://www.sciencedirect.com/science/article/pii/S1877050911002791, proceedings of the International Conference on Computational Science, ICCS 2011
16. Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (Apr 2009). https://doi.org/10.1145/1498765.1498785, http://doi.acm.org/10.1145/1498765.1498785
17. Xue, Q., Wang, Y., Zhan, Y., Chang, X.: An efficient gpu implementation for locating micro-seismic sources using 3d elastic wave time-reversal imaging. Computers & Geosciences **82**, 89–97 (2015)

18. Zhou, G., Zhang, X., Lang, Y., Bo, R., Jia, Y., Lin, J., Feng, Y.: A novel gpu-accelerated strategy for contingency screening of static security analysis. International Journal of Electrical Power & Energy Systems **83**, 33–39 (2016)
19. Zhou, J., Unat, D., Choi, D.J., Guest, C.C., Cui, Y.: Hands-on performance tuning of 3d finite difference earthquake simulation on gpu fermi chipset. Procedia Computer Science **9**, 976–985 (2012)