Seismic imaging on novel computer architectures

Jairo Panetta, Paulo Souza, Carlos Cunha, André Romanelli, Fernando Roxo, Ivan Pedrosa, Silvio Sinedino, Luiz Monnerat, Leandro Carneiro, Carlos Albrecht, TECNOLOGIA GEOFÍSICA, PETROBRAS, Brazil

This paper was prepared for presentation during the 11th International Congress of the Brazilian Geophysical Society held in Salvador, Brazil, August 24-28, 2009.

Abstract

This paper describes performance results obtained by porting production seismic imaging software to a set of novel processor architectures, with particular emphasis on a cluster of GPUs.

Introduction

Seismic processing algorithms are constantly evolving, in close connection with innovations in seismic data acquisition technology. In addition to that, an oil company has to adequate its proprietary algorithms to easily interrelate with a small set of commercial processing packages. In order to cope with this dynamic environment, production codes for seismic imaging developed at Tecnologia Geofísica of Petrobras are designed to accommodate multipurpose algorithms under a software structure that emphasizes portability and modularity.

But the lifetime of a software design may be limited by the lifetime of the computer architecture it was designed for. New trends in processor architecture (multi-core and many-core) and the ever-increasing number of processors on a cluster may trigger software redesign.

The flexibility of Petrobras seismic imaging software makes it a natural candidate to test the adequacy of a set of different algorithms and a previously designed software structure to the recently arrived computer architectures, since testing demands a limited amount of change in the actual production code.

This paper describes the structure of seismic imaging software developed at Petrobras, its use in daily production runs and the porting and achieved speed of one algorithm – the Kirchhoff time migration – to novel computer architectures, including a cluster of GPUs.

Seismic imaging software structure and capabilities

One of the main characteristics of the seismic imaging software developed at Petrobras is the structural flexibility to implement new functionalities. Some of these features are listed below:

- The same program can perform time migration, or time migration velocity analysis, in 2D or 3-D, pre-stack or post-stack, in one pass or two pass, as well as depth migration.
- The time migration and the migration velocity analysis can accommodate just a velocity field, or simultaneously a velocity field and an extra parameter field, which can be related either to anisotropy (VTI) or ray bending.
- The execution can originate from distinct commercial platforms (ProMax, Omega) or independently of any platform.

In some cases, in order to incorporate a new functionality the algorithm has to be redesigned to fit the main code structure, while seeking to reuse most of the previously developed modules. This is the case of the depth migration function, where only the traveltime computation module differs from the one-pass time migration code. To achieve this, the vertical axis of the traveltime tables is converted to vertical traveltime in such a way that the depth migrated output has also time as the vertical axis. Two positive side effects of this strategy are: regular sampling of the output (same as input) with all the anti-aliasing control defined in the time domain, and the fact that the output is already in the time domain, where most of the post conditioning algorithms (band pass filtering, for example) are to be applied before the subsequent conversion to depth. Once these post migration processing is completed, a simple vertical-ray depth conversion is necessary to produce the required depth volume.

This structural flexibility is a software design goal that may be in danger due to recent trends in computer architecture. It is well known that the lifetime of production code (decades as indicated by [1] and [4]) outlasts the computer architecture it was originally designed for. That is the case of our seismic imaging software, which suffered major reorganizations since its initial release (1995) to accommodate symmetric multi processors (SMP), clusters of SMPs, and distributed memory clusters with ever-increasing processor count. The newcomer multi-core and many-core processor architectures [2], [3] may change software design drastically. Current software characteristics highlights, detailed described bellow, are:

- Portability over computer platforms and commercial processing packages, with high performance on x86 processors;
- Flexible, dynamically balanced, fault tolerant MPI parallelism;

- Performance (and degree of parallelism) driven by a single parameter;
- Built-in execution time instrumentation automatically collected and stored for every run.

The application is divided in two main parts – a sequential one, which is incorporated into commercial processing packages, and a parallel code that runs on large clusters. The sequential part interacts with the user via the commercial platform API, and as so, it has to be cast to each package. It dynamically interacts with the parallel part by file exchange, sending input data and receiving computed results. The parallel part executes the seismic algorithm, and the same parallel code serves all packages.

Both parallel and sequential source code are written in standard conforming Fortran 95. The parallel source uses MPI-1.0 and has about 1K lines of C to speed-up IO.

To increase parallelism, domain decomposition occurs at the output space, using the fact that the computation of each output trace is independent of any other output trace. The output surface is partitioned into non-overlapping rectangular *blocks* of adjustable size. The block is the parallel unity of work (parallel grain).

A single master process schedules output block computations to slave processes by slaves demand. Dynamic load balancing accommodates uneven processing requirements and speeds.

Block size drives performance. For a fixed output area, larger block sizes limit parallelism but decreases re-reading and re-filtering of input traces.

The parallel kernel – compute an output block – is heavily vectorized and optimized. Effective portability (portability maintaining top speeds) over x86 based processors is achieved by the heavy use of performance analysis tools (such as [5], [6]). The kernel is composed by a triple nested loop. The innermost loop computes all contributions of an input trace to an output trace. The middle loop selects the target output trace from the given block. The outermost loop sweeps all input traces that may contribute to the output block, reading and filtering the selected trace.

Innermost loop memory references are the filtered input trace (load), the output trace (load, modify, store), the velocity (load) and scratch area. The middle loop brings another output trace and velocity to the inner loop. Since the filtered input trace fully dominates the inner loop memory requirements, the obvious cache reuse policy is to keep the filtered input trace in cache and tolerate cache misses for the output trace and velocity. This policy was implemented with careful coding, resulting in L2 usage of about 160KB and L2 cache hit ratio in excess of 99%. The outermost and the mid loops do not fit cache, since the average block requires about 100MB of memory.

The application has binary reproducibility – output file content is independent (binary) of processor count

Fault tolerance is central to long-running jobs on large Beowulf clusters. It is implemented by MPI intercommunicators, as suggested by [7]: each slave process communicates with the master process by a dedicated intercommunicator, allowing the master process survival when a slave crashes. In such a case, the master process reschedules the failed block to remaining slaves. Processes are statically assigned to processors by a one-to-one mapping.

Built-in execution time instrumentation generates performance data for every run, which is automatically stored at a data base. This performance data is central to drive performance developments, to correct user's choice of performance sensitive execution parameters (e.g. core count) and to detect machinery performance failures. The data base eases data retrieval and statistics computation over multiple jobs. It stores our own performance metric: the *number of contributions accumulated per second*, defined as the speed of input samples contributions to output samples.

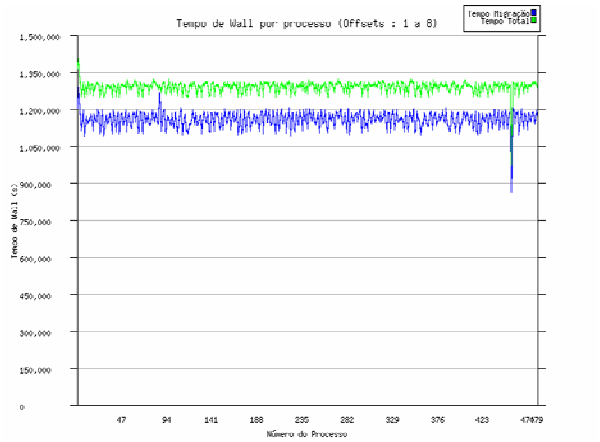

Figure 1: Wall clock distribution across processors

As an example of code capabilities and instrumentation, Figure 1 contains a typical production run total execution time and the migration loop execution time (both wall clock) across 500 processors. From the total execution time of about 1,300,000 seconds, about 1,150,000 seconds (88.4%) are used by the migration loop. Remaining execution time is used to filter input traces (about 10.2%) and IO. Plot jigger shows negligible load unbalancing. It also shows a faulty processor (numbered 453) decommissioned at about 900,000 seconds of run time.

Production spans a few thousand jobs per year. Table 1 contains the average core count and execution time of the 10, 100 and 1000 most demanding jobs over one year. Data shows that high processor count and long execution times are usual in production.

Table 1: Characterizing Production

	2006		2007	
Job count (top)	**Average core count**	**Days of execution**	**Average core count**	**Days of execution**
10	787	22	950	36
100	656	10	815	17
1000	185	2	196	3

As illustrated in Table 2 the number of jobs evolved from about 1600 in 2006 to about 3000 along 2008 and the top core count evolved from 1200 in 2006 to 3160 in 2008.

Table 2: Production Evolution

2006		2007		2008	
No. of Jobs	**Max no. of cores**	**No. of Jobs**	**Max no. of cores**	**No. of Jobs**	**Max no. of cores**
1565	1200	2685	2048	3000	3200

Reducing power consumption and heat dissipation have been the driving forces of recent computer architecture proposals. Massive parallelism is another clear trend. In this scenario of major changes, testing new architectural trends became of strategic importance. Petrobras have been testing these new trends since 2005. A previous work [8] describes our findings on the IBM BlueGene, multi-core x86 and the Cell Broadband Engine. A summary of these findings is described bellow, for the benefit of the reader. Research proceeded by porting the software structure to a GPU and to a cluster of GPUs, also described bellow.

Performance on large clusters of multi-cores

By late 2005, Petrobras and IBM cooperated on a research project to test the adequacy of the IBM Blue Gene/L [9] to the production Kirchhoff time migration application. A few available time slots on the four racks, 8192 processors IBM Blue Gene/L at IBM Rochester were granted to the project. Timing experiments were performed at Rochester to be compared with those obtained on a 1000 processor Beowulf blade cluster at Petrobras (2GHz Opteron 246 single-core processors). Table 3 compares speed/core (in mega contributions per second and core), speed/power (in mega contributions per second and watt) and speed/price (in kilo contributions per second and dollar) on both machines. *All the prices shown in this and other tables throughout this paper reflect the prices at the time the respective technology was introduced.*

The experiment is somehow unfair to the Blue Gene since the compiler does not vectorize single precision loops [14]. Even so, this pioneer architecture shows its strength in speed/watt and parallel scalability: the measured compound processing speed on a Blue Gene run with 8192 processors was 37.7G contributions per second against 29.3G contributions per second on the 1000 processors Beowulf cluster. The experiment also shows code scalability, with runs up to 8192 cores.

Table 3: Blue Gene and blade performance

System	**Cores/node**	**Speed/core**
Single core blade	2	29.30
Blue Gene/L	2	4.61
System	**Power/node (W)**	**Speed/Power**
Single core blade	153.67	0.38
Blue Gene/L	19.53	0.47
System	**Price/node (US$)**	**Speed/Price**
Single core blade	4000	14,65
Blue Gene/L	2000	4.61

Over recent years, this same application was tested on a set of x86 different nodes. These include old dual processor servers with two single-core 1.8GHz Opteron 244 (named Slow Single Core), dual processor servers with two single-core 3.06GHz Intel Xeon (a top sample of the high frequency, high power consumption architectural trend, named Fast Single Core), dual processors blades with two 2GHz dual-core Opteron 270 (a pioneer dual-core, named Dual core blade) and a prototype desktop machine with two 2.66GHz quad-core Intel Xeon (named Quad core). The same code was submitted to all different nodes, using processor specific compilation switches. Table 4 summarizes the results, using Table 3 nomenclature and units.

The speed per core improvements with the number of cores per chip show that the computational characteristics of the application suited multi-cores quite well. The high cache hit ratio avoids competition for memory. The low cache footprint and process independency avoid competition for cache lines among cores. The high computational intensity and high vectorization ratio enables simultaneous floating-point vector operations to be dispatched, utilizing recent multi-core micro-architecture enhancements.

Table 4: Multi core performance

System	**Cores/node**	**Speed/core**
Slow Single Core	2	30.24
Fast Single Core	2	36.67
Dual core blade	4	33.98
Quad core	8	49.01
System	**Power/node (W)**	**Speed/Power**
Slow Single Core	209.61	0.29
Fast Single Core	238.16	0.31
Dual core blade	198.94	0.68
Quad core	415.13	0,94
System	**Price/node (US$)**	**Speed/Price**
Slow Single Core	2500	24.19
Fast Single Core	2500	29.34
Dual core blade	4000	33.98
Quad core	6300	62.23

Speed per power and speed per price also increase with cores per chip, leveraged by the speed per core improvements. To identify the gains, it suffices to normalize the three ratios by the slow single core values. Figure 2 shows speed per power and speed per price gains in excess of speed per core gains.

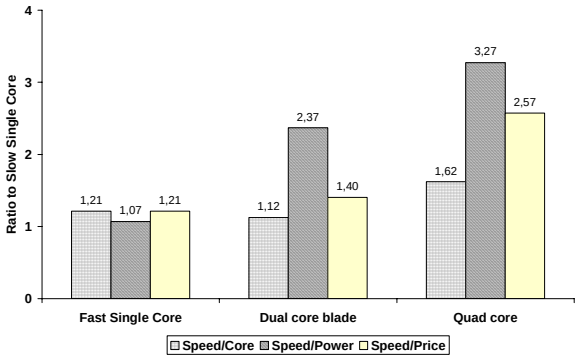

Figure 2: Gain ratios to slow single core

Cell Broadband Engine

Performance results presented in the last section were obtained without any source code modification. Consequently, the software design survived the multi-core trend at current core per chip count. But the many-core trend, represented by the Cell and GPU, requires kernel rewriting to explore the architecture full potential.

The Cell Broadband Engine Processor Architecture [10] is a promising heterogeneous architecture that achieves an interesting combination of outstanding floating point processing speed and low power consumption by using multiple simple cores and relinquishing memory consistency. It contains eight simple but powerful cores (SPE) and a PowerPC core (PPE). Each SPE controls its own flat memory – a critical 256KB – while the PPE has a regular memory size and hierarchy. Data is moved among memories by explicit DMA requests over a high bandwidth internal bus. Data movement and coherency among memories are the programmer's responsibility.

The Cell processor is the central unit of the gaming console Sony PlayStation 3 (PS3). PS3 attracts by its low cost (US$ 600 at the time of its introduction) and mass production, even being powered by a stripped down Cell processor (only 6 SPEs are available for general processing). In 2007 Petrobras build a modest cluster with four PS3 to test the adequacy of this technology to the application.

The parallelism strategy within a single Cell was to assign to each SPE the computation of all contributions of a single input trace to a single output trace and to assign to the PPE the reading and filtering of input traces. Given a filtered input trace, each SPE issues DMA requests of output traces and input velocities, copying data from the PPE memory to its own memory. Each SPE updates the output trace with the input trace contributions and issue a DMA request to store the updated output trace on PPE memory. The set of SPEs splits the output block, and output traces are transferred to each SPE by SPE demand. The PPE and the SPEs synchronize at a barrier whenever an input trace was fully processed. Double buffers at the PPE memory allow pipelining of input trace reading and filtering, minimizing barrier waiting time. Multiple Cells use the untouched master – slave parallelism, where the master process runs on a dedicated Cell and assigns output blocks to slave Cells by slave demand.

SPE memory space and DMA timing are critical issues to the success of this strategy. Each SPE uses about 160KB of memory to compute all contributions of an input trace to an output trace (the used x86 L2 cache space). Remaining SPE memory is used for double buffers of output trace and velocity, to allow pipelining of hardware serialized DMA requests. The high flop count per load/store (*computational intensity*) allows enough time for DMA completion for up to eight SPEs, even at high SPE floating-point arithmetic processing speeds.

The first PS3 experiment investigates parallel scalability within a single Cell, varying the number of SPEs involved in the computation. Figure 3 presents speed-up and parallel efficiency computed from wall clock times of the master process that runs on a second, dedicated PS3.

Execution times scale quite well with SPE count, showing algorithm adequacy to the Cell.

The second PS3 experiment repeats the multi-core experiment on a single slave PS3, driven by a dedicated master PS3. Table 5 summarizes the results, using the same nomenclature and units as before and considering one Cell SPE as a core. The quad-core results of Table 4 are repeated to ease comparisons.

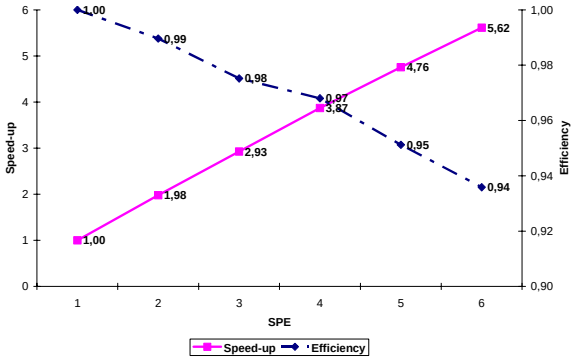

Figure 3: Single PS3 parallel performance

Table 5 shows that the PS3 exceeds in all metrics. A full PS3 is about 25% faster than a full quad-core. The PS3 is the most power efficient of all nodes, and has at least one order of magnitude better price/performance ratio over all conventional technology tested in these experiment..

Table 5: Single PS3 performance

System	Cores/node	Speed/core
Quad core	8	49.01
PS3	6	81.38
System	**Power/node (W)**	**Speed/Power**
Quad core	415.13	0,94
PS3	380.73	1.28
System	**Cost/node (US$)**	**Speed/Price**
Quad core	6300	62.23
PS3	600	813.80

The third PS3 experiment measures parallel scalability on the PS3 cluster, using the multi-core experiment input data and a dedicated master PS3.

Table 6: PS3 cluster parallel performance

Slaves PS3	2	3
Speed-up	1.99	2.92
Efficiency	0.99	0.97

The measurements reported by Table 6 show adequate parallel scalability of the time migration on the PS3 Cluster, although limited by the modest cluster size.

Graphics Processing Units

Graphics processing units (GPUs) are auxiliary devices specialized on graphics operations, usually coupled to x86 CPU boards or within a game console. They are particularly interesting to HPC applications due to a potentially very high floating-point execution speed at a modest price. The lack of a programming model and data transfer ratios from CPU memory to GPU memory and back, as well as the modest GPU memory size have been major impediments, over the years, to the adoption of GPUs in HPC. Recently released GPUs have overcome some of these issues.

Current GPUs dedicated to HPC such as the NVIDIA Tesla C1060 can be seen as a set of 30 processors, clocked at 1.3 GHz, that oversee a large (4GB) graphics memory ([12], [13]). Since bringing data from the graphics memory to the GPU processor takes 400 to 600[13] cycles, these cycles may be used by the processor to perform other, independent computations. In order to do that, each processor receives a set of independent threads and interleaves execution of instructions of different threads. Best performance occurs when the set of available threads perform identical operations (such as on vector operations – see [11] for details).

Porting our application to this architecture requires rewriting its kernel. The GPU computation starts with the output block and requires the velocity field to reside in graphics memory. A filtered input trace is dispatched to the graphics memory. The set of 30 processors compute all contributions of the input trace to the output volume at the rate of one contribution per thread, using enormous amount of fine grain parallelism. While the GPU computes all contributions of an input trace, the companion x86 CPU reads and filters the next input trace. CPU and GPU synchronize whenever the next input trace is filtered and the current input trace is fully processed. At this point, the CPU delivers the new, filtered, input trace to the GPU memory.

We recoded the kernel for the GPU using CUDA [13] and conducted execution time experiments on a research cluster of 36 quad-core, dual processor 2,33GHz Xeon nodes, accelerated by two NVIDIA Tesla C1060 GPUs per node, recently (late 2008) acquired by Petrobras. Table 7 contrasts its performance with previously tested machines, naming a “core” each of the 30 GPUs processors.

Table 7: Single GPU performance

System	**Cores/node**	**Speed/core**
Quad core	8	49.01
PS3	6	81.38
GPU	60	136.09
System	**Power/node (W)**	**Speed/Power**
Quad core	415.13	0.94
PS3	380.73	1.28
GPU	734.94	11.11
System	**Cost/node (US$)**	**Speed/Price**
Quad core	6300	62.23
PS3	600	813.80
GPU	10000	816.54

Results are astonishing, mainly when considering speed/power and speed/node and realizing that the GPU is an attachment to x86 nodes. The attachment costs around US$ 1850 per GPU (increase of 60%) plus the increase in power consumption (around 80%), but it speeds up the computation by a factor of 17. The incremental add-on to a well established commercial product is a major quality of this disruptive technology. On the other hand, we should point out that not all x86 nodes are able to accommodate GPUs, especially because the GPUs should be installed in fast PCIe Gen2 slots in order to avoid bottlenecks to the data transfers.

Table 8 contains the speed-up with the increase on the number of nodes, showing that the scalability of the software design is maintained.

Table 8: GPU cluster parallel performance

Slaves GPU	2	18	34	70
Speed-up	2.00	17.87	33.19	67.81
Efficiency	1.00	0.99	0.98	0.97

We observe a very efficient speed-up within the present capacity of our GPU cluster. Velocity field reading at the beginning of output block processing, and output data writing at the end compromises less than 1% of total wall time. Trace reading takes 1.3% of total wall time, but since this time overlaps with GPU computation, it does not affect performance at all.

In another test we increased CPU clock speed, CPU bus speed and CPU main memory speed by a factor not less than 20% in each item, the improved performance in our application was less than 1% showing that this application is GPU bound.

Conclusions

This paper probes the adequacy of the seismic imaging production software design to new trends in computer architecture. It describes design characteristics that were critical to the success of up to 3200 processors, or 36 days production runs: algorithm organization, sequential efficiency, parallel scalability, dynamic load balance, and fault tolerance. It presents performance data on a variety of x86 processor technologies, from old single-core to recent quad-core nodes, as well as Blue Gene/L performance data and two samples of the many-core trend, the Sony PlayStation 3 and NVIDIA GPUs. Or, from another angle, experimental data allows sensing the adequacy of processor technology trends to the time migration algorithm. Probed trends are multi-core and many-core.

Our measured data shows that the multi-core trend at current core per chip count is a viable substitute for the higher frequency single-core trend in any tested performance metric for time migration. It also shows that the many-core disruptive technology, represented by the Cell-based PS3 and by GPU attachments to x86 nodes, has the best performance in any metric, including an order of magnitude price/performance gain over any mass production system tested. But these gains have a major

cost – rewriting the application kernel to unleash the processing speed of these novel computer architectures.

Our current software design survived quite well to the novel architectures. It kept its characteristics of accommodating multiple algorithms without any change to the user's perspective. It also kept its macroscopic computational properties of scalability, dynamic load balance and fault tolerance. So far, adaptations to new computing environments were limited to its kernel, provided that the new environments are balanced.

Porting the application kernel to GPU took 15 days against 90 days to PS3. To achieve high performance in CUDA programming is relatively easy. Due to the scalar approach, vectorization (including *if* statements) is transparent to developer, once the data has been copied to GPU memory (4GB) its access is pretty much the same as in regular C programming and the huge number of simultaneous threads automatically hides pipeline and memory latencies. On the other side, Cell programming requires the developer to manage explicit vectorization (with no vector masked instructions), memory access control of small pieces of data (less than 256KB) (e.g. double buffering using DMA) to hide memory latency, and explicit loop unrolling to hide pipeline latencies. Besides, code and data (including buffers) must fit 256KB, which is SPE memory capacity. For example, 25 seismic traces with 3000 samples will not fit in the SPE memory.

Future work comprises measuring the performance of the depth migration on available machinery, as well as testing the resiliency of each solution.

Acknowledgments

The authors thank Petrobras for the long term research opportunity and for authorizing the public dissemination of research results. The authors gratefully acknowledge the continuous support of AMD, Intel, NVIDIA and IBM, including the early availability of prototype boards for performance measures. Blue Gene experiments would not be possible without the continuous support of Fabio Gandour and Marcelo L. Braunstein of IBM Brazil as well as José E. Moreira of IBM USA, among others.

References

[1] D. E. Post and L. G. Votta, "Computational Science Demands a New Paradigm", *Physics Today* 58(3), pp 35-41, Jan 2005.

[2] H. Sutter, J. Larus, "Software and the Concurrency Revolution", *ACM Queue*, Sep 2005.

[3] J. Dongarra, D. Gannon, G. Fox and K. Kenedy, "The Impact of Multicore on Computational Science Software", *CTWatch Quaterly* 3(1), Feb 2007.

[4] D. E. Post and R. P. Kendall, "Software Project Management and Quality Engineering Practices for Complex, Coupled Multiphysics, Massively Parallel Computational Simulations: Lessons Learned from ASCI", *International Journal of High Performance Computing Applications* 18(4), Dec 2004.

[5] OProfile, available at http://oprofile.sourceforge.net.

[6] S. Browne, J. J. Dongarra, N. Garner, G. Ho and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors", *The International Journal of High Performance Computing Applications*, 14(3), pp. 189-204, Mar 2000.

[7] W. Gropp and E. Lusk, "Fault Tolerance in Message Passing Interface Programs", *The International Journal of High Performance Computing Applications*, 18(8), pp. 363-372, Aug 2004.

[8] J. Panetta, P. R. P. Souza Filho, C. A. Cunha Filho, F. M. Roxo da Mota, S. S. Pinheiro, I. Pedrosa Jr, A. L. Romanelli Rosa, L. R. Monnerat, L. T. Carneiro and C. H. B. Albrecht, "Computational Characteristics of Production Seismic Migration and its Performance on Novel Processor Architectures", *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, IEEE Computer Society, 2007.

[9] G. L. Chui, M. Gupta and A. K. Royyuru, Guest Editors, "Blue Gene", *IBM Journal of Research and Development,* 49(2), pp. 189-500, May 2005.

[10] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer and D. Shippy, "Introduction to the Cell Multiprocessor", *IBM Journal of Research and Development,* 49(4), pp 589-604, Jul 2005.

[11] V. Volkov and J. W. Demmel, "Benchmarking GPUs to Tune Dense Linear Algebra", *SC'08,* Nov 2008.

[12] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, SS. S. Stone, D. B. Kirk and W. W. Hwu, "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA", *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ACM Press, 2008.

[13] NVIDIA CUDA Programming Guide, available at http://www.nvidia.com/object/cuda_develop.html.

[14] "Exploiting the Dual FPU in Blue Gene", available at http://www-03.ibm.com/systems/resources/systems deepcomputing pdf exploitingbluegenedoublefpu.pdf, page 3.