GPU Accelerated Stochastic Inversion of Deep Water Seismic Data

Tomás Ferreirinha[1], Rúben Nunes[2],
Amílcar Soares[2], Frederico Pratas[1], Pedro Tomás[1], and Nuno Roma[1]

[1] INESC-ID / IST, Universidade de Lisboa, Portugal
[2] CERENA / IST, Universidade de Lisboa, Portugal

Abstract. Seismic inversion algorithms have been playing a key role in the characterization of oil and gas reservoirs, where a high accuracy is often required to support the decision about the optimal well locations. Since these algorithms usually rely on computer simulations that generate, process and store significant amounts of data, their usage is often limited by their long execution times. In fact, the acceleration of these algorithms allows not only a faster execution, but also the development of larger and more accurate models of the subsurface. This paper proposes a novel parallelization approach of a state of art Stochastic Seismic Amplitude versus Offset Inversion algorithm, by using heterogeneous computing platforms based on a unified OpenCL programming framework. To take full advantage of the computational power made available by systems composed by multiple (and possibly different) accelerators, a spatial division of the simulation space is performed, enabling the parallel simulation of multiple regions of the geological model. This allows achieving a performance speed-up of 22.8× using two distinct GPUs without compromising the accuracy of the obtained models.

Keywords: Stochastic Inversion of Seismic Data, Heterogeneous computing, Graphics Processing Unit (GPU), OpenCL.

1 Introduction

In the last few years, High Performance Computing (HPC) platforms have been playing a key role in the Oil and Gas prospecting industry. As a result of the increased computing capabilities that have been offered to this industry, complex computational models of the subsurface can now be applied to estimate reserves and to diagnose and improve the performance of oil and gas producing fields. However, such applications are characterized by huge execution times (up to months), limiting their usefulness in most cases.

Stochastic algorithms have an important role in the characterization of oil and gas reservoirs, where accurate predictions are essential and the available information is often scarce and expensive. To compensate this issue, complex geological interpretations are made by using approximate computational models that simulate the oil reservoirs. Some related stochastic algorithms have

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part I, LNCS 8805, pp. 239–250, 2014.

already been optimized and parallelized in multi-core General Purpose Processors (GPPs) [1–3]. Some promising results were also obtained by using Graphics Processing Units (GPUs), such as the parallel implementation of the Stochastic Simulation with Patterns (SIMPAT) algorithm [4].

However, typical approximations result in models with a high level of uncertainty, leading to a faulty understanding of the geological structure and consequently to drilling errors. The recently proposed stochastic seismic Amplitude Versus Offset (AVO) inversion algorithm [5] using the Direct Sequential Simulation (DSS) [6] approach represents a promising methodology to solve geophysical inversion problems. It improves the generated models at a cost of a significantly more complex processing of the gathered data, with strict non deterministic dependencies among the several operations.

To the best of the authors' knowledge, only one parallel implementation of the DSS algorithm was proposed [3], where a multi-core approach was implemented by considering a straightforward functional decomposition of the algorithm, presenting considerable limitations in terms of scalability.

This paper proposes a parallelization approach of the stochastic seismic AVO inversion algorithm by considering heterogeneous platforms, composed by several devices with different computational capabilities. To achieve such a flexible solution, the proposed implementation uses the OpenCL API, allowing each part of the algorithm to be easily migrated among the several coexisting GPPs and GPUs. After a careful analysis of the characteristics of the algorithm, it was verified that the most significant part of the algorithm is composed by millions of dependent iterations that individually are not computationally demanding. Due to the lack of data parallelism opportunities presented, a relaxation of the algorithm was considered in order to efficiently exploit the highly parallel architecture of such platforms, significantly reducing the algorithm execution time without compromising the quality of the obtained models.

2 Stochastic Seismic AVO Inversion

The main goal of seismic inversion problems is to estimate a set of models that characterize the physical properties of the Earth subsurface, given a limited set of observed measurements. The reservoir models that are generated via stochastic inversion algorithms can be significantly improved with the integration of different kinds of information [7], *e.g.* well log and seismic reflection data. The most common methodology to incorporate this seismic information, in stochastic fine grid models, is known as geostatistical inversion [8]. Here, a state of art Stochastic Seismic AVO Inversion algorithm [5] is considered, which is an iterative method, based on the Global Stochastic Inversion [7, 9] approach. This method directly inverts the density (ρ), P-wave velocity (Vp) and S-wave velocity (Vs) models, allowing the use of AVO analysis, which can not be done with the more common acoustic inversion methods. The stochastic seismic AVO inversion method can be summarized in the following steps, which are also illustrated in Figure 1:

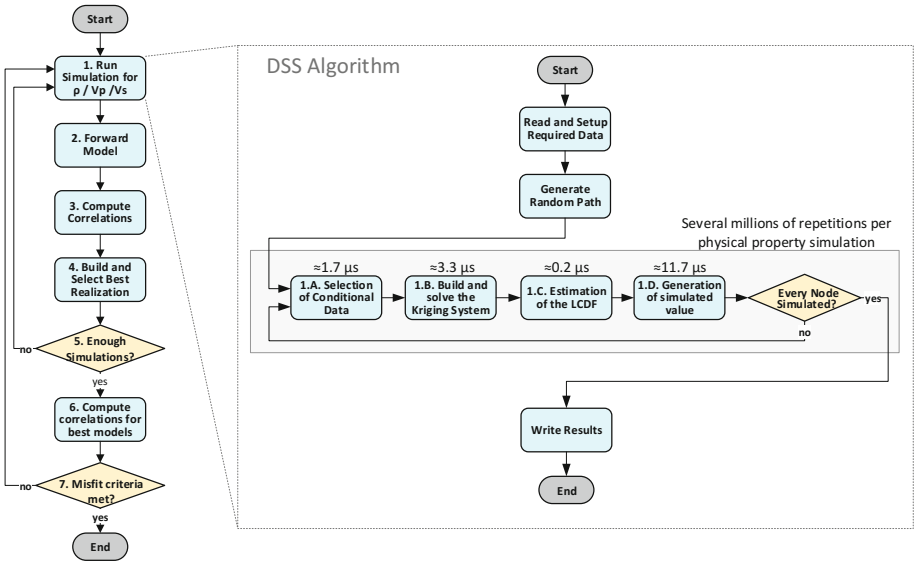


Fig. 1. Stochastic seismic AVO inversion algorithm flowchart. The highlighted procedure corresponds to the most computational demanding part of the whole algorithm.

1. Stochastic simulation of the ρ, Vp and Vs models, by using the DSS with joint probability distributions algorithm;
2. Calculation of the synthetic pre-stack seismic cube with the simulated ρ, Vp and Vs models, by using Shuey's approximation [5];
3. Comparison between the synthetic seismic cube and the real seismic data, creating a correlation cube that evaluates multiple regions of the model;
4. Definition of the best ρ, Vp and Vs models, using a genetic based algorithm;
5. Repeat steps 1 to 4 until no more realizations are desired;
6. Creation of the correlation cubes regarding the best ρ, Vp and Vs models;
7. Repeat of the whole procedure, using the best models and respective correlation cubes to condition the next generation of simulations.

The stochastic simulation (step 1) adopts a Direct Sequential Simulation and Cosimulation [6] approach, which starts by defining a random path through a grid of nodes, to be considered during the simulation process. The stochastic nature of the algorithm depends on this random path in order to find multiple equiprobable models, and consequently to converge. Afterwards, each node is simulated at a time, conditioned by the real data and all the previously simulated values. The simulation procedure can be summarized in the following steps:

A. Randomly select a node from a regular grid;
B. Construct and solve a kriging system for the selected node;
C. Estimate a Local Cumulative Distribution Function (LCDF) at the selected node, by linear interpolating both the real and the experimental data available in the neighbourhood (kriging estimate);

D. Draw a value from the estimated LCDF, using a Monte Carlo method;
E. Return to the step A, until all nodes have been visited by the random path.

The cosimulation variant of this algorithm enables the simulated variable to be conditioned to other previously simulated variables, without any prior transformation. In this case it is only used for the simulation of Vp (conditioned by the previously simulated density model) and Vs (conditioned by the previously simulated Vp model). This is one of the main advantages of the DSS algorithm, when compared with the other sequential simulation algorithms, as is the case of the Sequential Indicator Simulation (SIS) and Sequential Gaussian Simulation (SGS) algorithms.

3 Parallelization Approach

3.1 Problem Analysis

As stated by Amdahl's law, only the most demanding and time-consuming sections of an application are worth parallelizing. Accordingly, the algorithm was profiled to find the most time consuming phases, considering two distinct datasets: a smaller one composed of a grid of 101x101x90 nodes, and a larger and more realistic one with 237x197x350 nodes. For such purpose, the execution time of the different parts of the algorithm was accurately measured using the Performance Application Programming Interface (PAPI) [10] to interface with the hardware performance counters. From the obtained profiling results, it was verified that more than 90% of the algorithm execution time is spent in the generation of the ρ/Vp/Vs models. Consequently, this part of the algorithm was chosen as the prime focus for acceleration. In addition, other performance counters were used in order to evaluate the limiting factor of the algorithm performance. It was verified that the application has a significant amount of memory instructions per floating point operation, which indicates that the application is mainly memory bounded. The same conclusion can be drawn by using SchedMon [11] to analyse the Cache-Aware Roofline model [12], where it is clearly observed that the execution samples are located in the memory bounded region of the model (see Figure 2).

The subsequent study considered a comprehensive analysis of the several existing data dependencies in the generation of the ρ/Vp/Vs models. In particular, as represented in Figure 3(a), nodes from a not strictly defined neighbourhood of the node being simulated are selected as conditioning data. As represented in Figure 3(b), the estimation of the local conditional distribution function (step C) requires the values of the previously simulated nodes (step D). As a consequence, although steps A and B can be parallelized, the algorithm execution path is limited by the sequential random path through every node, composed by steps C and D, which still represents approximately 85% of the execution time. Thus, by only performing the first two steps in parallel, it will be hard to significantly accelerate the execution. Finally, individually accelerating each step of the algorithm would also be difficult, since the complexity of the algorithm

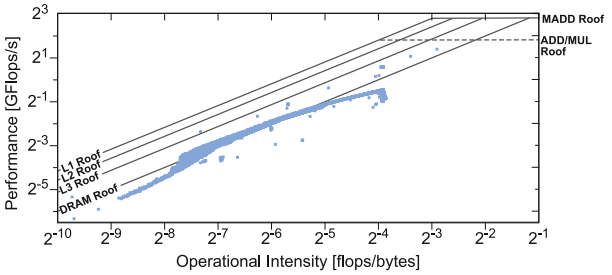
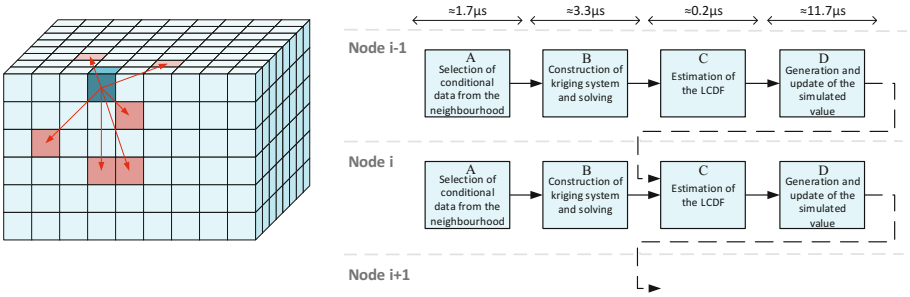


Fig. 2. Roofline model of the sequential execution of the DSS algorithm. Each point in the chart represents a 10ms sample of the program execution.



(a) Conditional data locations. (b) Example with 3 nodes.

Fig. 3. Data dependencies on the stochastic simulation of the ρ, Vp and Vs models

lies in the number of nodes that are required to be simulated (usually several millions) and not in the simulation of a single node, that not only presents few parallelization opportunities but also does not take enough time to be worth the overhead of transferring data to other devices.

3.2 Parallelization Approach

To circumvent the limitation imposed by the described data dependencies, the proposed approach is based on a relaxation of this problem by dividing the simulation grid in multiple tri-dimensional sub-grids, and then randomly selecting a node per sub-grid. Therefore, at each step of the simulation procedure, a set of nodes are simulated and updated at once, conditioning the subsequent nodes to be simulated (see Figure 4). Along the simulation, every sub-grid will select the nodes to be simulated through the same relative sub-path, thus granting a constant distance between the nodes being simulated in parallel. Note that the considered sub-divisions of the simulation grid are not required to be cubic. In fact, the anisotropic nature of the seismic data being processed indicates that there are significantly less dependencies in the vertical direction, which allows for a greater vertical division of the simulation grid.

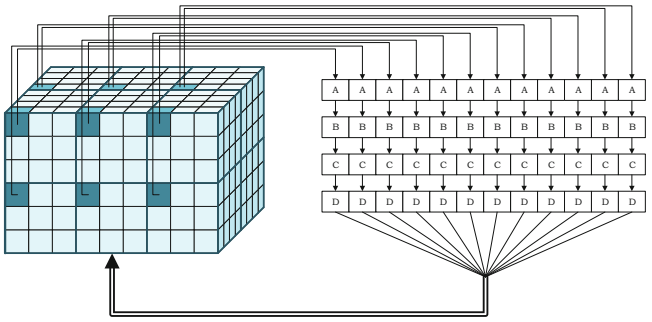


Fig. 4. Parallelization approach based on a spatial division of the original grid

Since the kriging estimate is computed by considering the closest available data to the node being simulated, a spatial sub-division of the simulation grid guarantees algorithm convergence, as it will be seen in section 4. In fact, as long as the nodes being simulated are sufficiently apart from each other, being kept outside of the search range, no data conflicts should be observed [13]. This may become a limiting factor for smaller datasets, which provide less opportunities to split the grid in many pieces without avoiding such conflicts in the first iterations of the simulation procedure, where few data is available and not uniformly distributed over the simulation grid.

Considering the GPU architecture and the adopted OpenCL programming framework, there are at least two different ways to map the proposed approach: each sub-grid is simulated by a distinct OpenCL work-group, being the inherent parallelism of the algorithm explored by the OpenCL work-items within each work-group; each sub-grid is simulated by a distinct work-item. The main difference regarding both approaches is concerned with the number of nodes being simulated at the same time, since in the former approach more resources are being assigned to the simulation of a single node, thus limiting the number of nodes that can actually be simulated at the same time. However, those resources are only completely used when the code itself is parallelizable by a multiple of the warp/wavefront size, which is not the case for a significant part of the algorithm, where only a single work-item in the work-group would effectively be performing useful computations. Although the second approach allows for a greater amount of nodes to be simulated in parallel, it is more memory demanding (intermediate buffers need to be replicated for every node under simulation), and performance losses may occur due to warp divergence and non-coalesced memory accesses, since each thread is simulating its own node, which may lead to different execution paths or accesses to different regions of the memory. In case that the device memory available is not enough, the simulation procedure is performed by simulating the grid layer by layer, keeping a complete copy of the grid being simulated only in the host device. This layers must contemplate not only the blocks that will be simulated but also the neighbour blocks that will condition the simulation.

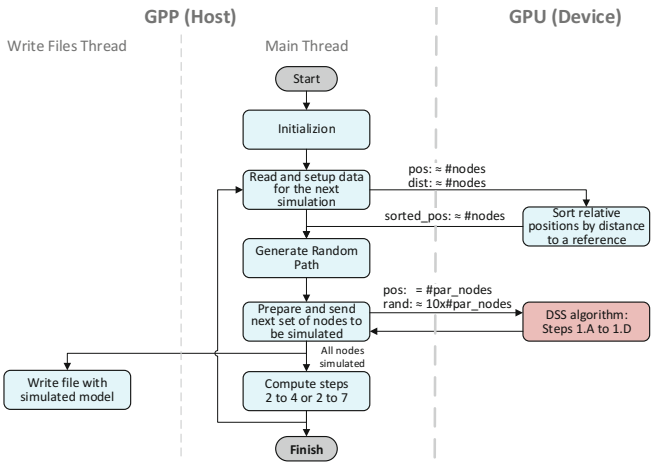


Fig. 5. Execution flowchart of the algorithm. The DSS Algorithm being executed in the GPU is the most computational demanding part.

Accordingly, as illustrated in Figure 5, only tasks related with the simulation procedure are effectively being performed by the GPU devices, being the host device responsible not only by the generation and sending of the data required in every step of the simulation procedure (overlapped with the computations), such as the nodes to be simulated in parallel in the following step or the random values required during the procedure, but also by performing all the other not so significant parts of the algorithm.

3.3 Considered Optimizations

Optimization of the DSS Algorithm Kernels. Several optimizations were considered in order to optimize the performance of the proposed implementation. A significant improvement comes with the efficient usage of local memory in order to optimize not only parallel reductions, which are required to compute the mean and variance of the previously simulated nodes in order to compute the probability distribution functions, but also to improve the access times to global memory buffers that are frequently used. Another aspect that in some cases may be significant is the overhead related with the OpenCL calls. In fact, considering the current AMD GPU drivers, those overheads are in the order of hundreds of microseconds, which may become significant if there are a considerable number of kernel calls or memory transfers per iteration. At this respect, the data transfers were performed using the minimum number of OpenCL enqueue buffer calls possible, and a balance was achieved between the size of the kernels, which influences the number of registers being used and consequently limits the occupation of the device, and the number of kernel calls, which is related not only with the number of different kernels but also with the number of divisions that are performed to the simulation grid. The used data structures and its

indexing were also optimized to increase the coalescence of memory accesses by the work-items of a given work-group.

Application Specific Optimizations. Some other application specific optimizations were also considered, such as the use of a bottom-up merge sort algorithm in order to sort multiple arrays in parallel, instead of using the quick-sort that provided better sequential results. This happens because the merge sort algorithm has a fixed execution path if the arrays being sorted have the same size, thus leading to significantly less warp divergence. Also, given the characteristics of the algorithm under study, a significant part of the execution time lies in the first steps of the simulation procedure, most specifically in the conditional data search. This happens because the search procedure starts by looking for available data from the closest nodes relative to the node being simulated, until a given number of conditioning nodes is found (defined as a parameter). Therefore, since initially the available data is scarce and it is not evenly distributed in the simulation grid, the simulation of nodes in regions with few available data results in significantly larger execution times. This problem was minimized by postponing the simulation of nodes from blocks in which there is not enough available data both in the block itself and in the neighbour blocks. This optimization can be performed during the definition of the random path, since it is only required to know the number of available data per block in each iteration. As a result, the execution time of the first steps is significantly reduced at the cost of some extra steps in the end of the simulation algorithm, when there is already a large amount of available data, thus reducing the global execution time.

Optimizations to the Rest of the Algorithm. Finally, some optimizations outside of the simulation procedure were also performed, such as the use of the bitonic sort algorithm in order optimize the sorting of the array that stores the nearest relative positions to a given reference, according to the non-euclidean distances between nodes (related with the anisotropy of the data). Moreover, the output files that have to be written during the execution of the algorithm are also written in parallel by a different GPP thread, thus becoming overlapped with the computations, in order to avoid as much overhead as possible.

3.4 Multi-device Approach

Another important aspect that should also be considered is concerned with the possibility of using multiple devices, in order to ensure another level of scalability. A scalable approach was thus devised to decrease the execution time when multiple GPU and GPP devices are available. Under such condition, the proposed approach starts by first dividing the complete grid between the OpenCL enabled accelerators, and then by sub-partitioning the sub-grids into smaller blocks which are computed in parallel in each device. Also, to efficiently exploit different computational capabilities delivered by different devices, the load is balanced by distributing the nodes to be simulated between the multiple devices

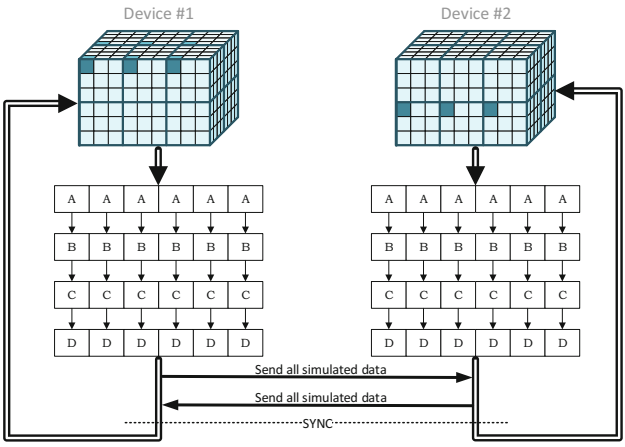


Fig. 6. Multi-device parallelization approach

according to real-time performance measurements. Nevertheless, since the simulation grid is stored in the device memory, it must be updated after the parallel simulation of every bundle of nodes with the information being computed in the other devices. This implies an all-to-all communication scheme and consequently a synchronization point. In the end of the simulation procedure, the resulting model must be read from any device, since every device has an updated copy of the simulation grid (see Figure 6).

4 Experimental Results

The experimental setup considered the execution of 5 iterations of the algorithm, composed by 8 sets of simulations each, over two real datasets: one with 101x101x90 and other with 237x197x350 nodes. Performance was measured by comparing the execution times obtained using multiple heterogeneous environments with the sequential execution of the algorithm in an Intel i7-3820 processor, using the -O3 compiler optimization flag.

Figure 7 shows the obtained performance results when considering several distinct mappings (programmed with the same OpenCL source code). It must be noted that the simulation of the different physical properties (ρ, Vp and Vs) uses different data along the simulation procedure, which naturally introduces some variations in the resulting execution times (18%, 17% and 65% of the simulation execution time respectively). Also, only the time during the simulation procedure itself (effectively being accelerated) was considered for those speed-up measurements, without considering the time required to setup the simulation data (5% of the sequential simulation execution time).

From the obtained results, it can be observed that the execution time was significantly reduced in all the considered mappings. Namely, a speed-up of 15.8× was obtained, considering the execution of the whole algorithm using a single

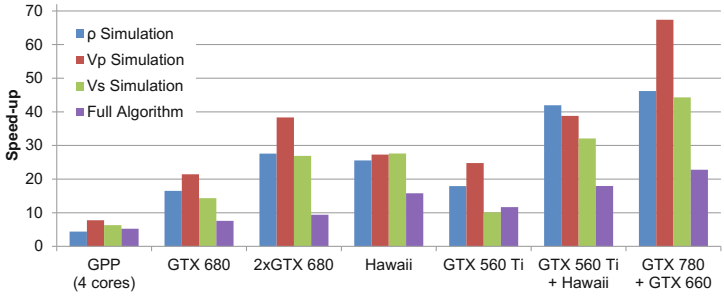
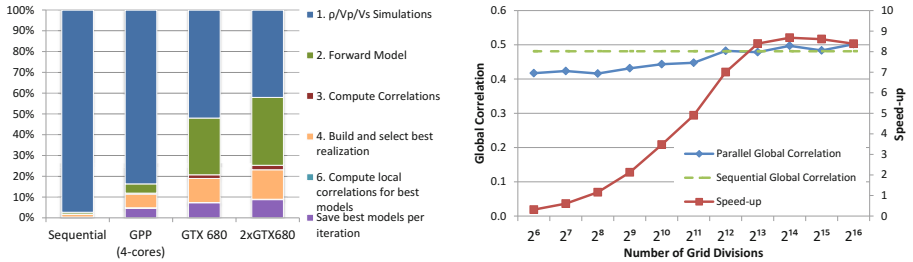


Fig. 7. Execution speed-up results compared to the sequential GPP implementation, when considering different heterogeneous environments, using the dataset with 237x197x350 nodes

GPU, and performance improvement of 5.3× when the algorithm was mapped into the GPP. In the latter case, the obtained speed-up is bigger than 4 (number of cores) due to the use of the hyper-threading technology. The observed performance improvements are coherent with previously mentioned limitations of the sequential implementation. In fact, by comparing the effective bandwidth of the sequential implementation (using the L1/L2/L3 cache hit rates), with the Hawaii GPU theoretical bandwidth, a speed-up of 15× was expected. The speed-ups observed experimentally were greater than this theoretical value because the algorithm was optimized in order to efficiently use the shared memory, and the computational parts of the algorithm are also being parallelized (e.g. parallel reductions), further improving the algorithm performance.

Regarding the multi-device approach, when considering two GTX 680 GPUs, higher improvements were verified in the simulation procedures (up to $\simeq$1.8× when comparing with a single device execution). However, only a global speed-up of 1.24× was obtained because, as the performance is being improved, this procedure becomes less significant in the overall execution time (see Figure 8(a)). Significant improvements were also verified when considering the cooperative execution with multiple different devices, which demonstrates the scalability of the considered implementation in heterogeneous environments. Namely, by using devices from different manufacturers with different computational capabilities, a global speed-up of 18× was obtained.

It must be noted that a slightly worse scalability was verified in the less computational demanding simulations (ρ and Vp), because the non-coalesced memory accesses of the conditional data search procedure, due to the parallel access to different regions of the grid, becomes a significant limiting factor. Also, the machine using GTX 680 GPUs uses a different host device (Xeon E5-2609), which justifies the observed full algorithm speed-up differences. Nevertheless, although the obtained multiple device speed-up is slightly below the theoretical limit (mainly due to the communication overhead), the load-balancer successfully managed to divide the work-load between the available devices, maximizing the achieved performance.



(a) Functional workload distribution, using the 237x197x350 dataset.

(b) Convergence and speed-up analysis, using the 101x101x90 dataset.

Fig. 8. Graphical analysis of the obtained results comparing with the sequential implementation of the algorithm

Despite being memory demanding, this implementation occupied approximately 580 MB of the device memory which, considering modern devices, still leaves room for a significant scaling of the problem size until a layer by layer simulation approach is required (that implies a slightly increased communication overhead).

Finally, Figure 8(b) presents the evolution of both the speed-up and the global correlation coefficient with the number of grid-divisions (which corresponds to the nodes being simulated in parallel). The results were obtained by performing several independent runs of 10 iterations, using the Hawaii GPU and considering the dataset composed by 101x101x90 nodes. As it can be observed, when comparing the simulated models with real data, the convergence is still verified, since the obtained global correlation coefficients are similar both for the parallel and sequential implementations of the algorithm. This demonstrates that the applied relaxations, toward an efficient parallelization, do not affect the quality of the results. In fact, it can be verified that, even when a very significant amount of sub-grids was considered (resulting in a simulation composed only by 15 steps), the convergence of the algorithm was still verified. This is mainly due to the postponing optimization that avoids simulating nodes that have few or no conditional data in the neighbour blocks. As a result, when the block size becomes smaller (i.e., the number of grid divisions increases), a consistent spatial distribution of the physical property being simulated is granted, which improves the algorithm convergence.

5 Conclusions

This paper proposes a parallelization of a state of the art stochastic seismic AVO inversion algorithm in heterogeneous platforms. The acceleration of such algorithms not only allows for faster reservoir modeling, but also to make it possible to develop larger and more accurate computational models of the Earth's subsurface. To circumvent the strict data dependencies presented by this algorithm,

the adopted approach considers a spatial relaxation of the dependencies and consequent division of the simulation grid, thus allowing the parallel simulation of multiple nodes corresponding to different regions of the model. Such division comes with no loss of accuracy in the results. According to the obtained experimental results, the proposed acceleration efficiently balances the work-load between multiple (possibly different) devices, achieving speed-ups over 22× in the heterogeneous configuration with two different NVIDIA GPUs.

Acknowledgment. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), under projects: Threads (PTDC/EEA-ELC/117329/2010), P2HCS (PTDC/EEI-ELC/3152/2012) and project PEst-OE/EEI/LA0021/2013.

References

1. Stinessen, B.O.: Profiling, Optimization and Parallelization of a Seismic Inversion Code. PhD thesis, Norwegian University of Science and Technology (2011)
2. Hysing, A.D.: Parallel Seismic Inversion for Shared Memory Systems. PhD thesis, Norwegian University of Science and Technology (2010)
3. Nunes, R., Almeida, J.A.: Parallelization of sequential gaussian, indicator and direct simulation algorithms. Computers & Geosciences 36(8), 1042–1052 (2010)
4. Tahmasebi, P., Sahimi, M., Mariethoz, G., Hezarkhani, A.: Accelerating geostatistical simulations using graphics processing units (GPU). Computers & Geosciences 46, 51–59 (2012)
5. Azevedo, L., Nunes, R., Soares, A., Neto, G.: Stochastic seismic AVO inversion. In: 75th EAGE Conference & Exhibition Incorporating SPE EUROPEC 2013 (2013)
6. Soares, A.: Direct sequential simulation and cosimulation. Mathematical Geology 33(8), 911–926 (2001)
7. Caetano, H.M.V.: Integration of seismic information in reservoir models: Global Stochastic Inversion. PhD thesis, Universidade Técnica de Lisboa (2009)
8. Haas, A., Dubrule, O.: Geostatistical inversion-a sequential method of stochastic reservoir modelling constrained by seismic data. First Break 12(11) (1994)
9. Soares, A., Diet, J., Guerreiro, L.: Stochastic inversion with a global perturbation method. In: EAGE Petroleum Geostatistics (2007)
10. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A portable programming interface for performance evaluation on modern processors. International Journal of High Performance Computing Applications 14(3), 189–204 (2000)
11. Taniça, L., Ilic, A., Toms, P., Sousa, L.: Schedmon: A performance and energy monitoring tool for modern multi-cores. In: 7th International Workshop on Multi/many-Core Computing Systems, MuCoCus 2014 (2014)
12. Ilic, A., Pratas, F., Sousa, L.: Cache-aware roofline model: Upgrading the loft. IEEE Computer Architecture Letters (2013)
13. Vargas, H., Caetano, H., Filipe, M.: Parallelization of sequential simulation procedures. In: EAGE Petroleum Geostatistics (2007)