Computers & Geosciences 78 (2015) 26–36

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Acceleration of stochastic seismic inversion in OpenCL-based heterogeneous platforms ☆

Tomás Ferreirinha [a], Rúben Nunes [b], Leonardo Azevedo [b], Amílcar Soares [b], Frederico Pratas [a], Pedro Tomás [a], Nuno Roma [a,*]

[a] INESC-ID/IST, Universidade de Lisboa, Rua Alves Redol, 1000-029 Lisboa, Portugal
[b] CERENA/IST, Universidade de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 9 August 2014
Received in revised form
19 January 2015
Accepted 6 February 2015
Available online 11 February 2015

Keywords:
Stochastic inversion of seismic data
Heterogeneous computing
Graphics processing unit (GPU)
OpenCL

ABSTRACT

Seismic inversion is an established approach to model the geophysical characteristics of oil and gas reservoirs, being one of the basis of the decision making process in the oil&gas exploration industry. However, the required accuracy levels can only be attained by dealing and processing significant amounts of data, often leading to consequently long execution times. To overcome this issue and to allow the development of larger and higher resolution elastic models of the subsurface, a novel parallelization approach is herein proposed targeting the exploitation of GPU-based heterogeneous systems based on a unified OpenCL programming framework, to accelerate a state of art Stochastic Seismic Amplitude versus Offset Inversion algorithm. To increase the parallelization opportunities while ensuring model fidelity, the proposed approach is based on a careful and selective relaxation of some spatial dependencies. Furthermore, to take into consideration the heterogeneity of modern computing systems, usually composed of several and different accelerating devices, multi-device parallelization strategies are also proposed. When executed in a dual-GPU system, the proposed approach allows reducing the execution time in up to 30 times, without compromising the quality of the obtained models.

1. Introduction

In the last few decades, high-performance computing infrastructures have become essential tools in the oil and gas industry, in order to satisfy the ever growing needs for oil and gas production and exploration. However, due to the amount of data that is generated, processed and stored, applications used in this industry are often characterized by huge execution times (up to months), often limiting their usefulness whenever faster and high resolution models are required. To decrease the computation time, many solutions have been devised, including the development of specialized processing structures and the exploitation of modern computing devices both at the level of a single computing platform or at the level of a cluster or grid.

Among the multiple application domains, stochastic algorithms play a key role in the characterization of oil and gas reservoirs, where accurate predictions are required even in the presence of scarce information available. Accordingly, complex geological interpretations of the subsurface properties are usually attained by using approximate computational models. However, these approximations often result in subsurface models with a high level of uncertainty, leading to a faulty understanding of the geological structure and to subsequent drilling errors, with rather expensive consequences not only in terms of investment costs but also in terms of the environmental impact. The recently proposed stochastic seismic Amplitude Versus Offset (AVO) inversion algorithm (Azevedo et al., 2013), based on a Direct Sequential Simulation (DSS) (Soares, 2001) approach represents a promising methodology to solve geophysical inversion problems, improving the generated models at the cost of a significantly more complex processing of the gathered data.

To reduce the execution time, a simplified version of the DSS algorithm was parallelized to exploit multi-core Central Processing Units (CPUs) (Nunes and Almeida, 2010). By applying a straightforward functional decomposition of the algorithm, presented a multi-core implementation by considering a straightforward functional decomposition of the algorithm, an acceleration of up to 3.5 × was observed for a quad-core CPU. However, this solution is limited not only in scalability but also in the capacity to exploit modern heterogeneous computing systems composed of multiple processors.

The generic parallelization of geostatistical simulation methods have also been studied (Mariethoz, 2010), where three distinct

☆The source code of the proposed algorithm implementation is available at: http://sips.inesc-id.pt/tools/avocl

* Corresponding author. Fax: +351 213145843.
E-mail address: Nuno.Roma@inesc-id.pt (N. Roma).

http://dx.doi.org/10.1016/j.cageo.2015.02.005
0098-3004/

levels of parallelization were identified: *realization level*, *path level* and *node level*. In such broader scope, some related stochastic algorithms have already been optimized and parallelized using multi-core CPUs (Nunes and Almeida, 2010; Mariethoz, 2010; Stinessen, 2011; Hysing, 2010) and Graphics Processing Units (GPUs) (Tahmasebi et al., 2012). All these implementations presented different approaches to maximize the parallelization of the algorithm, while trying to avoid as much communication overhead as possible. However, whenever highly parallel environments are considered, non-fully distributed master–slave approaches are still prominent. In these cases, the master node often becomes a performance bottleneck as the number of parallel processing threads increases, since it has to satisfy the requests of every slave node. Furthermore, the increasing communication overhead between the multiple slaves and the master node can only be hidden by allowing asynchronous communications, which often prevents an exact reproducibility of the results.

Accordingly, an efficient parallelization approach of the Stochastic Seismic AVO Inversion algorithm is now proposed, by considering highly heterogeneous platforms composed of several devices with different computational capabilities. Such a flexible solution is achieved by using the OpenCL API, allowing each part of the algorithm to be easily migrated among the several coexisting CPUs and GPUs. Namely, the OpenCL framework is not restricted to a given vendor of GPUs neither to GPU architectures, offering more code portability and flexibility when compared to the CUDA programming language. It is up to the programmer to decide how efficient and how portable its implementation is going to be. Therefore, the proposed parallelization approach stands out from the current state of the art in the following aspects:

- Improved scalability, by executing the stochastic simulation procedure in a completely distributed processing scenario.
- Guaranteed reproducibility of the attained results.
- Efficient and scalable implementation in highly heterogeneous processing environments.

This paper is organized as follows. A contextualization and a brief description of the algorithm being parallelized is presented in Section 2. The proposed parallelization approach is described in detail in Section 3, serving as a background to the following section where the single and multi-device implementations are described. An experimental evaluation of the developed implementations is presented in Section 5. Finally, Section 6 concludes this paper, summarizing the main conclusions and contributions.

2. Stochastic seismic AVO inversion

Seismic reflection data plays a key role in hydrocarbon reservoir geo-modelling workflows. Numerical three-dimensional subsurface Earth models, built from available well-log and seismic reflection data, are usually considered essential tools for reliable decision making and risk analysis (Doyen, 2007). A geostatistical (or stochastic) framework allows the integration, within the same model, of a broad set of data with very different scale support, namely well-log data, which is sparsely located along the study area but has a very high vertical resolution; and seismic reflection data, which covers a great spatial extent but has a low vertical resolution (Bosch et al., 2010). Unfortunately, the integration of the seismic reflection data is not straightforward within the modelling workflow, since it is an indirect measurement of the subsurface elastic properties of interest, such as P-wave velocity (Vp), S-wave velocity (Vs) and density (ρ). Before this data can be integrated, the so-called seismic inverse problem needs to be solved (Bosch et al., 2010). However, due to measurement errors and approximations in the physical models that are often used to solve the seismic inverse problem, this is an ill-posed and highly nonlinear problem with a non-unique solution (Tarantola, 2005; Bosch et al., 2010).

High quality pre-stack seismic data with high signal-to-noise ratio and with a considerably high fold number is becoming the standard in exploration and characterization projects. For this reason, there has been an increase on the number of available inverse techniques for pre-stack seismic reflection data (e.g., Mallick, 1995; Buland and Omre, 2003). At the same time, due to the advantages on uncertainty assessment and data integration, geostatistical inverse procedures are increasing its popularity in seismic reservoir characterization studies. Accordingly, a state of art geostatistical pre-stack seismic inversion methodology (Azevedo et al., 2013, 2014) that allows the inversion of pre-stack seismic data directly for ρ, Vp and Vs models is herein adopted. Contrary to most inversion procedures that simply invert the seismic reflection data for acoustic and elastic impedance, this particular inversion methodology is able to individually invert each of the elastic properties of interest.

The adopted methodology consists of an iterative geostatistical inverse procedure based on two main parts: a genetic algorithm that acts as a global optimizer to ensure the convergence of the solution from iteration to iteration; and a stochastic sequential simulation (Direct Sequential Simulation (DSS), co-DSS (Soares, 2001) and co-DSS with joint probabilities distribution (Horta and Soares, 2010). This stochastic sequential simulation algorithm allows the calculation of a set of subsurface Earth models, each called a realization, that honour the experimental data (available well-log data); the prior probability distribution (or a target one) estimated from the well-log data; and a spatial continuity pattern, as revealed by a covariance model (e.g., in two-point geostatistics a variogram). The iterative geostatistical pre-stack seismic inversion methodology is summarized in the following modules (see Fig. 1):

- Joint simulation of density, P-wave and S-wave velocities:
  1. Stochastic simulation of r density models conditioned to the available well-log data with DSS (Soares, 2001).
  2. Stochastic co-simulation of r Vp models given the r previously simulated density models with co-DSS with joint-distributions (Horta and Soares, 2010).
  3. Stochastic co-simulation of r Vs models given the r previously simulated Vp models with co-DSS with joint-distributions (Horta and Soares, 2010).
- Optimization for the iterative convergence:
  1. From the triplet of elastic models computed in the previous stage, calculation of r angle-dependent synthetic pre-stack seismic volumes using Shueys linear approximation (Shuey, 1985).
  2. Comparison between each synthetic angle gather with the corresponding real angle gather on a trace-by-trace basis; and calculation of r local correlation gathers for each location within the seismic grid.
  3. Selection of the areas with the highest correlation coefficient, by simultaneously considering all angles to build the best density, P-wave and S-wave models; these best models are then used as secondary variables for the co-simulation of the elastic models generated during the next iteration.
  4. Iterate from (1) until the matching criteria (i.e., global correlation between the original and the synthetic pre-stack seismic) are reached.

Each stochastic simulation procedure (blocks 1, 2 and 3 in Fig. 1) can be summarized with the flowchart presented in Fig. 2, composed of the following processing blocks:

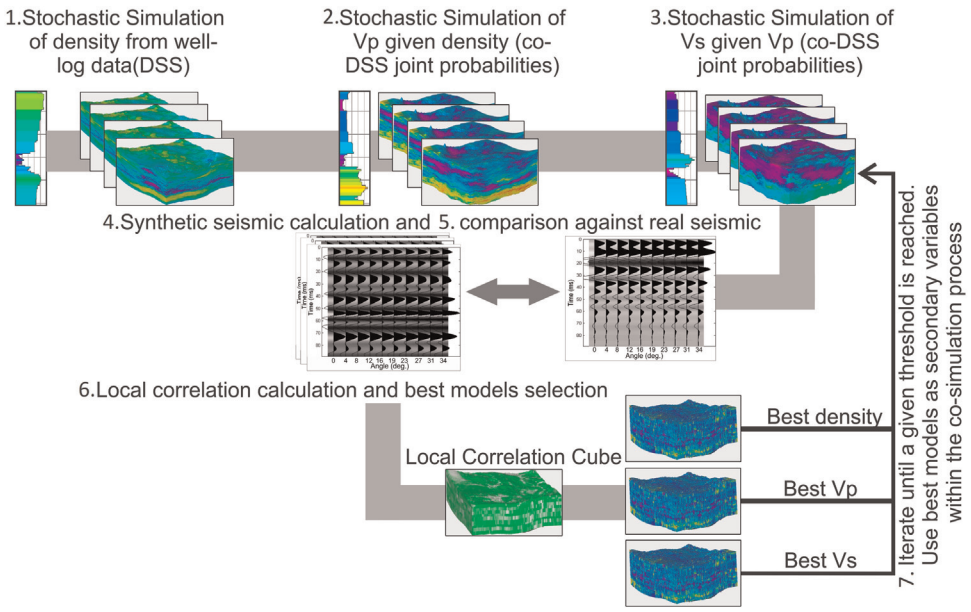

Fig. 1. Stochastic seismic AVO inversion algorithm flowchart.

(A) Randomly selection of a node from a regular grid.
(B) Construction and solution of a kriging system for the selected node.
(C) Estimation of a Local Cumulative Distribution Function (LCDF) at the selected node, by linear interpolating both the experimental data and previously simulated nodes available in the neighbourhood (kriging estimate).
(D) Drawing of a value from the estimated LCDF, by using a Monte Carlo method.
(E) Repeat from A, until all nodes have been visited by the random path.

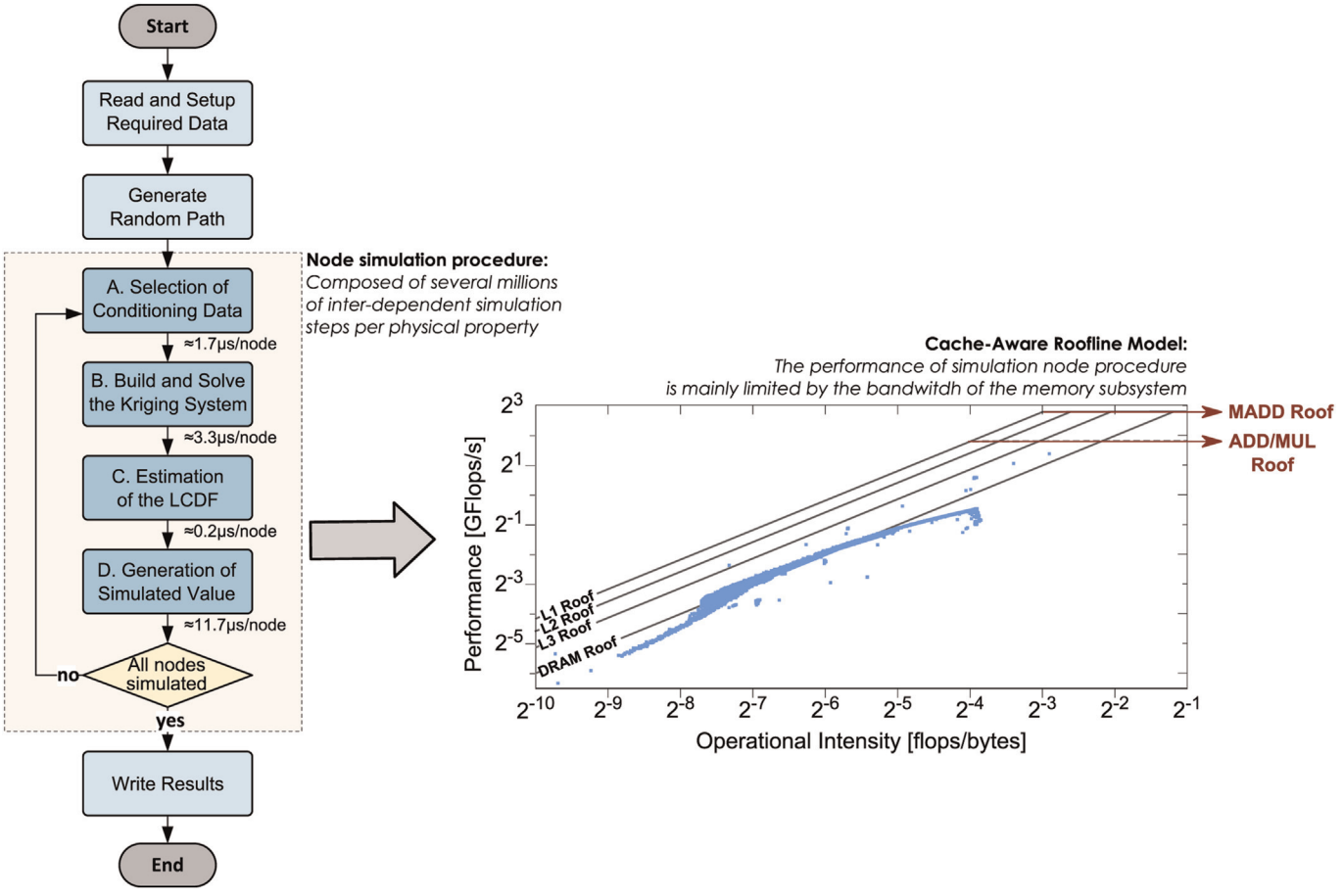

Fig. 2. DSS algorithm flowchart. The most computational demanding procedure is highlighted, together with the percentage of the execution time on each of the composing parts.

The co-simulation variant of this algorithm enables the simulated variable $Z_2(x)$ to be conditioned by other previously simulated variable $Z_1(x)$, without any prior transformation of the secondary variable. In fact, the conditional cumulative distribution function $F[Z_2(x)|Z_1(x)]$ is computed from the bi-distribution $F[Z_1(x), Z_2(x)]$, by using moving classes (Horta and Soares, 2010). When compared with the other sequential simulation algorithms, such as the Sequential Indicator Simulation (SIS) and the Sequential Gaussian Simulation (SGS) algorithms, this is one of the main advantages of the adopted DSS algorithm. Since the node visiting path is random, different runs will produce different models and consequently each conditioning data at a specific grid node may slightly differ from each other (Doyen, 2007).

Altogether, this iterative procedure ensures an accurate reproduction of the individual marginal distributions of each elastic property, as estimated from the well-log data, as well as the joint distributions between density and P-wave velocity and P-wave and S-wave velocities. All the simulated and co-simulated models generated during this iterative procedure also match the well-log data (at the wells locations) and the spatial continuity pattern, as revealed by a variogram model. At the end of this iterative and convergent methodology, synthetic pre-stack seismic data can be retrieved, with an accurate match in terms of reflectors position and amplitude variations with regard to recorded pre-stack seismic data. In addition, the corresponding ρ, Vp and Vs models can also be retrieved. This methodology was already successfully tested and implemented in a synthetic seismic dataset (Azevedo et al., 2013) and it is herein applied to the same challenging synthetic deep-offshore reservoir.

3. Algorithm parallelization

Aiming an efficient parallelization of the algorithm, an extensive analysis of its processing modules was conducted and is herein briefly described. Such analysis supports the subsequently proposed partitioning of the problem and the corresponding parallelization approach.

3.1. Problem analysis

In order to ensure an efficient partitioning and mapping of the algorithm into the considered parallelization platforms, a comprehensive analysis of its execution and of its implicit dependencies had to be made. For this purpose, a preliminary quantitative profiling analysis was conducted, in order to identify the most time consuming phases. Two distinct datasets were considered for this characterization: a small one, composed of a grid with $101 \times 101 \times 90$ nodes; and a larger and more realistic one, with $237 \times 197 \times 350$ nodes. As presented in Ferreirinha et al. (2014), the execution time of the different parts of the algorithm was carefully evaluated, being identified that the generation of the ρ/Vp/Vs models using the DSS algorithm corresponds to the most time consuming part (see blocks 1, 2 and 3 in Fig. 1). Furthermore, by relying on SchedMon (Taniça et al., 2014) to carefully monitor the algorithm execution on an Intel Core i7 3770K CPU, it was concluded that the application is mainly limited by bandwidth of the memory subsystem (see the Cache-Aware Roofline Model, Ilic et al., 2014, on the right side of Fig. 2).

The subsequent study considered an analysis of the several existing data dependencies of the DSS algorithm. At each step of the simulation procedure (processing sequence from parts A to D), conditioning data is selected by finding the k-nearest neighbouring nodes with available data (being the k value defined as a simulation parameter). Therefore, since the simulation procedure follows a random-path, conditioning data is selected from a not strictly defined neighbourhood of the node being simulated (see Fig. 3(a)). Accordingly, the estimation of the local conditional distribution function (part C) at each node makes use of values obtained from previously simulated nodes (part D), imposing a strict commitment of a set of data dependencies within the processing grid (see Fig. 3).

3.2. Parallelization approach

Taking into account the characteristics of the DSS algorithm (the most computationally demanding part of the inversion procedure), three distinct parallelization approaches can be considered: *functional-level*, by simultaneously executing multiple independent parts of the algorithm; *data-level*, where each part of the algorithm is individually accelerated by simultaneously processing independent data; *path-level*, where multiple nodes from the random path are simulated at the same time by different parallel processing threads.

Accordingly, by observing the set of existing dependencies between multiple parts of the algorithm (see Fig. 3(b)), it can be easily concluded that a *functional-level* parallelization would hardly provide good acceleration results. In fact, although parts A and B can be fully performed in parallel, the algorithm acceleration is limited by the sequential execution of the random path through every node (parts C and D), which still represents most of the simulation execution time. In particular, by applying Amdhal's Law to the acceleration of only parts A and B, a maximum speed-up of $1.4\times$ is attained. Furthermore, an individual acceleration of each part of the algorithm would also hardly give good results, not only because the multiple parts present few parallelization opportunities, but also because their sequential processing time is not enough to be worth the overhead of transferring data to other devices. In fact, the complexity of the algorithm lies in the several millions of nodes that have to be simulated and not in the

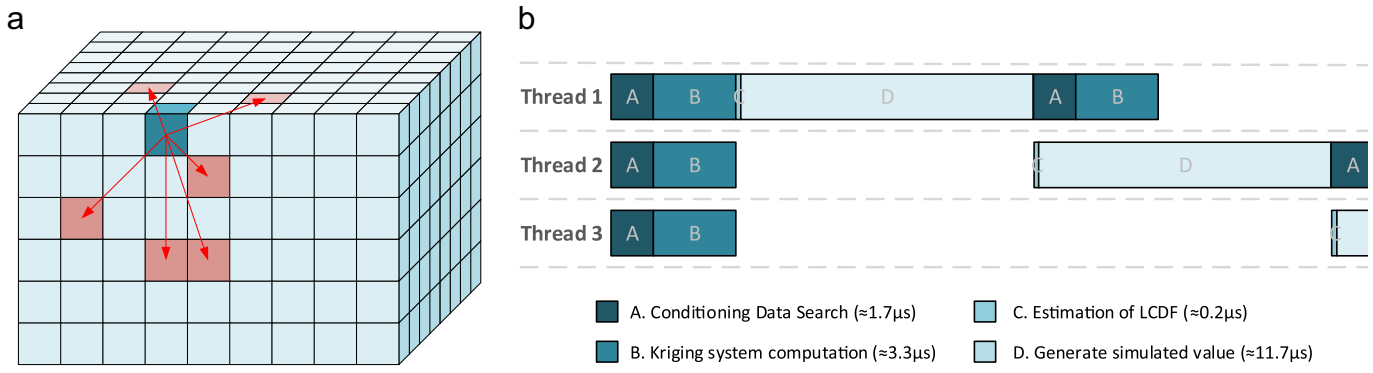

Fig. 3. Data dependencies on the stochastic simulation of the ρ, Vp and Vs models. (a) Conditioning data locations and (b) functional parallelization, by considering 3 threads.

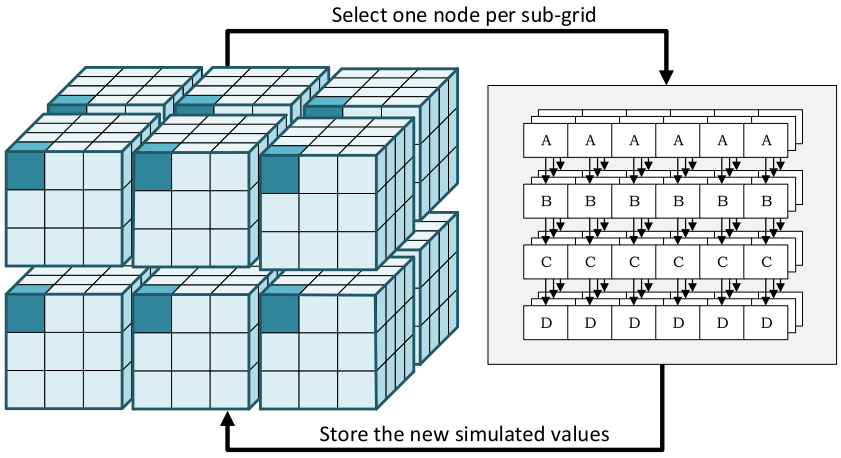

Fig. 4. Spatial division of the original simulation grid in multiple sub-grids.

simulation of a single node. Finally, in order to guarantee the same results as the original sequential version of the algorithm, a *path-level* parallelization has to ensure that no conflicts are observed between the nodes being simulated in parallel (neighbourhoods must not overlap), which introduces serious limitations in terms of scalability.

To circumvent these limitations, the proposed approach is based on a relaxation of the algorithm definition in order to enable an efficient (but still accurate) *path-level* parallelization. Such a parallel implementation was achieved by: (i) dividing the simulation grid in multiple sub-grids; and (ii) randomly selecting a node within each individual sub-grid at each step of the simulation procedure. At the end of each step, the whole set of nodes being simultaneously simulated is updated at once in the simulation grid, conditioning the subsequent simulation steps (see Fig. 4). At this respect, it is worth noting that the considered subdivisions of the simulation grid are not required to be cubic. In fact, the anisotropic nature of the seismic data being processed suggests that there are significantly less dependencies in the vertical direction, which allows for a greater vertical division of the simulation grid.

Regarding the generation of the random path, two distinct criteria can be considered to select the simulation nodes, namely: (i) by following the same relative sub-path in each sub-grid; or (ii) by allowing each sub-grid to have its own random sub-path. Taking into account that as long as the nodes being simulated are sufficiently apart from each other (being kept outside of the search ranges), no data conflicts should occur (Vargas et al., 2007), it can be observed that one of the main advantages of the first mode is that it grants a constant and equidistant distance between the nodes being simulated in parallel. In addition, such regularity in the set of nodes being simulated in parallel can also present some advantages in terms of the GPU's implementation efficiency. On the other hand, the second alternative presents less restrictions to the considered random path, increasing the number of possible paths and consequently the number of possible solutions (which is significant, considering the stochastic exploration-based nature of this algorithm). In this case, since no significant differences in terms of performance and of quality of the generated models were observed, the second alternative was chosen in order to provide a greater flexibility to the random path generation.

4. Exploiting GPU-based heterogeneous platforms

Taking into account the presented parallelization approach, the devised methodology to efficiently exploit heterogeneous platforms composed both of CPUs and GPUs is herein described. For such purpose, a brief introduction to GPU programming is presented, together with a detailed description of a single-device and multi-device approaches that allow exploiting both path-level and realization-level parallelism.

4.1. General-purpose GPUs programming

The GPU is a computer component originally designed to provide real-time 3D graphics rendering. In accordance, its architecture is significantly different from a CPU, such as to efficiently exploit data-level parallelism. Due to their highly parallel architecture, GPUs are nowadays widely used as general-purpose accelerators, for highly parallel computational demanding applications.

Despite being able to execute thousands of threads in parallel, modern GPU architectures have several constraints that may limit their performance. In particular, when an application is mapped into a GPU, the corresponding threads are grouped in warps, with all threads performing the same instruction at the same time (over different data). Furthermore, memory accesses may only occur in parallel provided that the memory positions that are accessed by the multiple threads of a given work-group obey to some specific restrictions in terms of the access patterns. As a consequence, since each computational GPU core is simpler than a CPU one, significant performance improvements are only verified when an application efficiently uses the GPU parallel architecture, demanding a specific optimization effort that frequently leads to a redesign of the algorithms being accelerated.

4.2. Single-device implementation

By taking into account the general constraints imposed by GPU architectures, allied with the adopted OpenCL programming framework, there are at least two different ways to map the proposed approach: (i) each sub-grid is simulated by a distinct OpenCL *work-group*, being the inherent parallelism of the algorithm exploited by the OpenCL work-items within each *work-group*; (ii) each sub-grid is simulated by a distinct *work-item*. However, since the simulation of a single node presents few parallelization opportunities, the former approach is not able to efficiently exploit the whole GPU architecture, which conducted to the selection of the second approach. Nevertheless, despite the greater amount of nodes that can be simulated in parallel with this approach, it is more memory demanding than the former, since intermediate buffers need to be replicated for every node under simulation. Hence, in the event that the available device memory is not

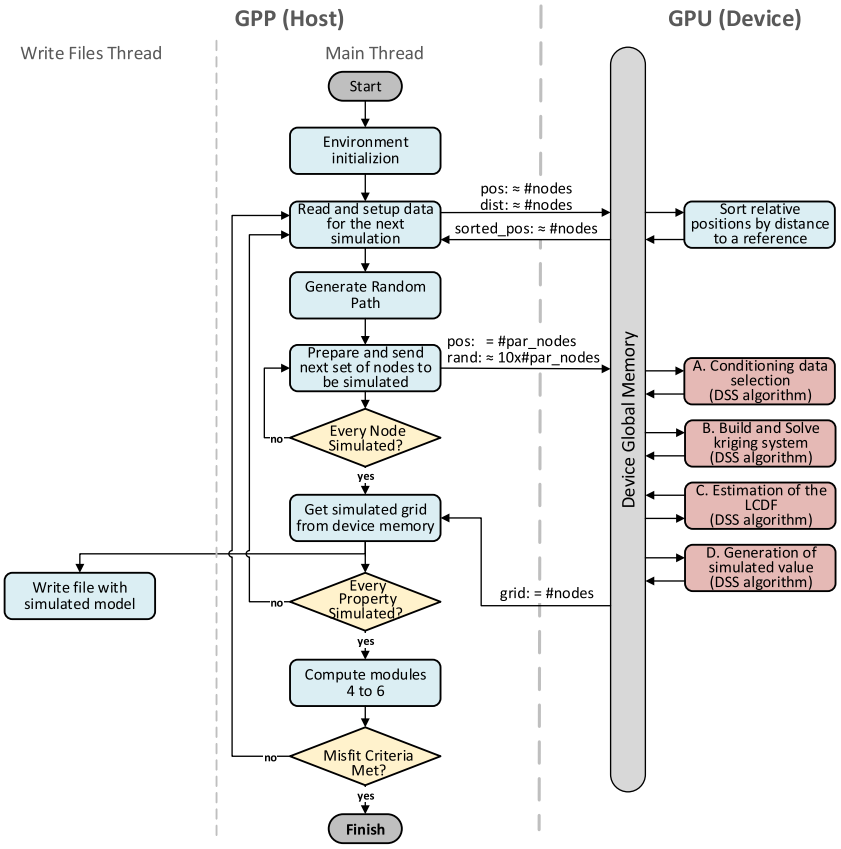

Fig. 5. Execution flowchart of the algorithm. The DSS algorithm being executed in the GPU (represented by blocks A, B, C and D, on the right) is the most computational demanding part.

enough, the simulation procedure can still be performed by simulating the grid layer by layer, keeping a complete copy of the grid being simulated only in the host device. These layers must contemplate not only the specific sub-grids that will be simulated, but also the neighbouring sub-grids that condition the simulation. Nevertheless, some performance losses may still occur, due to the different execution paths and memory accesses from threads of the same *work-group* to different regions of the memory, since each thread is simulating its own node.

It must be noted that only tasks related to the simulation procedure are actually performed by the GPU devices, being the host device responsible not only by the generation and transfer of the data required in the simulation procedure (overlapped with the computations), but also by the execution of all the other not so significant parts of the inversion algorithm (see Fig. 5).

4.3. Multi-device single-realization approach

Another important aspect that should also be considered is the possibility of using multiple devices, in order to ensure another level of scalability. Accordingly, the proposed approach takes advantage of the previously described spatial division of the simulation grid, and divides the obtained sub-grids between the several OpenCL enabled accelerators. For such purpose, the original simulation grid is initially divided into sub-grids, being the number of sub-divisions proportional to the number of available devices, in order to exploit the increased computational capabilities. The sub-grids are then distributed to the available accelerators according to its computational capabilities.

In order to efficiently take advantage of eventually different computational capabilities delivered by different devices, the load is balanced by dynamically distributing the nodes to be simulated between the multiple devices, according to real-time performance measurements obtained by OpenCL profiling events. At each step of the simulation procedure the sub-grids are re-distributed according to an iterative load balancing routine.

Since the simulation grid is stored in the device memory (one copy per device), it must be updated with the information being computed in the other devices after the parallel simulation of every bundle of nodes. This implies an all-to-all communication scheme and a consequent synchronization point (see Fig. 6). After receiving the whole set of new simulated values, each device updates (in parallel) its simulation grid, together with the mean and variance values of the already simulated nodes, by using parallel reductions. At the end of the simulation procedure, the resulting model may be read from any device, since every device has an updated copy of the simulation grid.

4.4. Multi-device multi-realization approach

Although significant improvements can be obtained by using multiple GPUs to accelerate a single realization, the attained scalability of such approach is limited by the intrinsic demand to communicate the simulated values upon each simulation step and by only parallelizing the execution of the simulation procedure. Accordingly, greater performance levels are expected by considering a parallelization scheme at the realization level, where several independent simulations are performed at the same time by multiple devices, thus minimizing the need for communications. In this approach, each device computes a different realization (set of ρ, Vp and Vs simulations), together with the corresponding forward model and the correlation related computations regarding those realizations (see Fig. 7).

Nevertheless, the construction of the best model still needs to

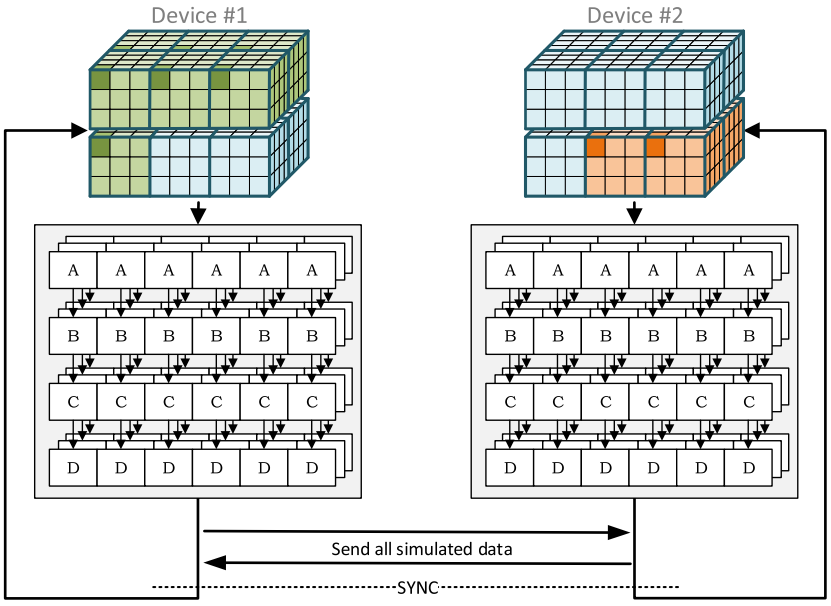

Fig. 6. Multi-device single-realization approach, by considering two devices with different computational capabilities.

be sequentially performed, since it selects the best regions from each generated model by comparing with the current best model. Accordingly, after executing a given realization, the synthetic seismic model must be processed by a CPU thread responsible for the best model construction. Hence, beyond the CPU thread dedicated to writing the output files, an extra host thread is used to build and select the best realization, and to build the local correlation cubes (main thread, in Fig. 7).

To maximize the performance, all devices iteratively simulate one realization without any midpoint synchronization. However, when a set of r realizations of the inversion algorithm have been performed, the local correlation cubes regarding the best models that were built have to be created, in order to condition the next generation of simulations. Hence, as soon as the last simulation of the current iteration is completed, all devices still under execution are interrupted (indicated with a cross, in Fig. 7), resuming the simulation of realizations once the best local correlation cube is constructed.

Accordingly, the multi-device multi-realization approach is only limited by the sequential execution of the best model construction procedure, which may only become a bottleneck if the number of devices significantly increases, and by the number of realizations per iteration, that limits the number of devices that can execute in parallel. Nevertheless, the first problem can be minimized by accelerating that specific part of the algorithm if deemed worthwhile, and the latter by considering the use of both the multi-device approaches (which may coexist), in order to provide another level of scalability to the algorithm execution.

4.5. Specific algorithmic optimizations

To improve the performance of the heterogeneous computing, several optimization procedures were implemented, as described in the following paragraphs.

Postponing of nodes with scarce pre-conditioning data: A significant part of the simulation execution time lies in the pre-conditioning search of the input data, during the first steps of the simulation procedure. The main reason for this overhead arises from scarceness and uneven distribution throughout the simulation grid of the input data, leading to significantly larger execution times when processing nodes from regions with few available data. To overcome this problem, the processing of nodes located in sub-grids where there is few available data (both in the sub-grid being simulated and in the neighbouring sub-grids) is postponed. This pre-conditioning can be performed during the definition of the random path, since, for each step of the simulation procedure,

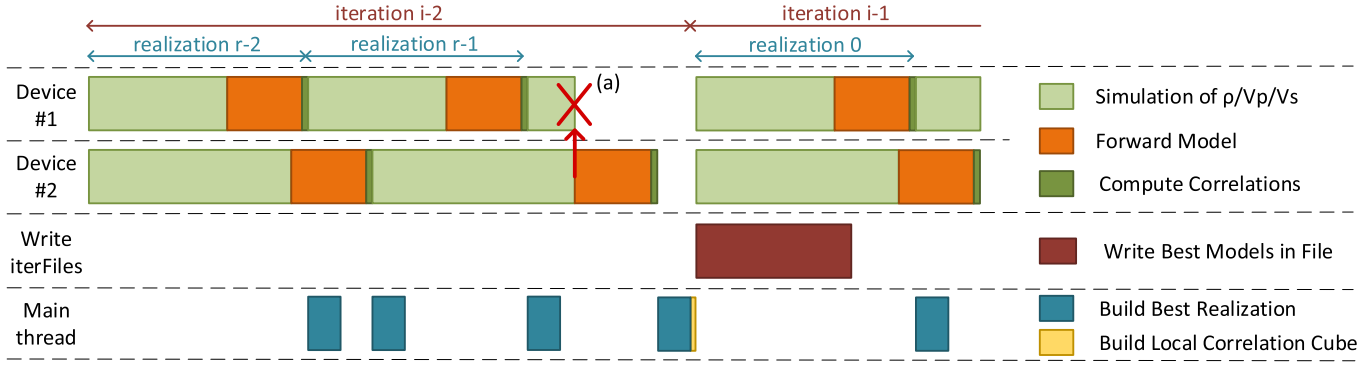

Fig. 7. Temporal diagram of the multi-device multi-realization approach, considering two devices with different computational capabilities and four realizations per iteration.

it is only required to know the number of nodes that will be available in each sub-grid, but not their exact positions. As a result, the execution time of the first steps can be significantly reduced at the cost of some extra steps at the end of the simulation procedure, when there is already a significant amount of available data, thus reducing the global execution time. A collateral benefit of this postponing method also arises in terms of the quality of the obtained results, since it avoids the simulation of nodes from regions with few data in a close neighbourhood (see Section 5.2).

Optimization of the sorting procedures: One of such optimizations refers to the usage of the bitonic sorting algorithm, in order to optimize the sorting of the array that stores the nearest relative positions to a given reference, according to the non-Euclidean distances between nodes (used to support the data pre-conditioning search procedure).

Efficient use of GPU local memory: A significant improvement comes from an efficient usage of the GPU local memory, not only to optimize parallel reductions, often required to compute the mean and variance of the previously simulated nodes, but also to reduce the mean access time to the frequently accessed global memory buffers.

Minimization of the OpenCL overheads: The reduction of the inherent parallelization overheads was tackled by performing the data transfers with a minimum number of OpenCL enqueue buffer calls. To attain this objective, a compromise was achieved between the granularity of the kernels, which influences the number of registers being used and consequently the occupation of the GPU, and the number of kernel calls, which is related both with the number of different kernels and with the number of divisions of the simulation grid. The inherent data structures, as well as its corresponding indexing, were also optimized in order to increase the coalescence of memory accesses and thus reduce overall execution overheads.

Overlapping of computation with the writing the results to file. In order to provide access to the set of best models at each iteration, these have to be written to a file at the end of each iteration. However, without careful considerations, this procedure can easily become a performance bottleneck. To avoid increasing the critical execution path, the writing of results to file is made in parallel with the GPU computations by relying on a different CPU thread.

5. Experimental results

The proposed parallelization was evaluated in terms of the attained performance of the algorithm execution and in terms of the quality of the inversion results, considering the adopted relaxation of the original algorithm. In the first case, the algorithm was evaluated by computing the experimental speed-up regarding an optimized serial implementation of the original algorithm. As for the analysis of the quality of the inversion results, the algorithm is herein evaluated not only in terms of its convergent behaviour, but also by comparing the obtained spatial distributions of the generated models.

5.1. Performance improvements

In order to measure the performance of the proposed parallelization approaches, the execution times using multiple heterogeneous environments (see Table 1) were compared with the execution of an optimized serial implementation of the algorithm in an Intel i7-3820 processor, by considering the -O3 compiler optimization flag, when executing 3 iterations, each composed of 10 realizations over the dataset with 237 × 197 × 350 nodes.

Fig. 8 shows the obtained performance results when considering several single device mappings (programmed with the same OpenCL source code). From the obtained results, it was observed that the execution time was significantly reduced in all the considered mappings. In particular, it was obtained a speed-up of 20.6 ×, when considering the execution of the whole algorithm using a GTX 780 Ti GPU, and a performance improvement of 5.9 × when the algorithm was mapped into the CPU. In the latter case, the obtained speed-up is even greater than 4 (number of CPU cores), due to the exploitation of the hyper-threading technology. It is also worth noticing that the multiple considered systems use different CPU devices, which justifies the full-algorithm speed-up discrepancies between systems.

Table 1
Considered systems specification details.

Component	System 1	System 2	System 3
CPU	Intel i7 3820	Intel Xeon E5-2609	Intel i7 4770K
RAM	16 GB	32 GB	32 GB
GPU 1	AMD Hawaii R9 290X	NVIDIA GeForce GTX 680	NVIDIA GeForce GTX780 Ti
GPU 2	AMD Pitcairn R7 265	NVIDIA GeForce GTX 680	NVIDIA GeForce GTX 660 Ti

When analysing the obtained results from the perspective of the multi-device approaches (see Fig. 9(a)), it can be observed that significantly higher throughputs were effectively attained when considering the multi-device multi-realization approach. Such already expected acceleration rate arises because: (i) a greater part of the algorithm is actually being executed in parallel; (ii) there is no penalization from communication and synchronization requirements during the whole simulation procedure; and (iii) the best model construction (computed in a host device dedicated thread) is being overlapped with computation of new realizations. Nevertheless, by evaluating the implementation from the point of view of the multiple device efficiency when accelerating the node simulation procedure (multi-device single-realization implementation), it was verified that the communication overhead corresponded to approximately 0.99%, 2.5% and 0.43% of the execution time for the ρ, Vp and Vs simulations, respectively. In addition, by analysing the efficiency of the implemented load balancing, it was observed global efficiencies of approximately 87%, 93% and 97% for the three types of simulations (ρ, Vp and Vs, respectively) when considering the execution over the two GPU devices of System 1. Similar load balancing efficiencies have also been verified when considering the remaining systems.

In what concerns the execution over distinct datasets, it has been verified that considerable accelerations were achieved in all cases, ranging from 9 × to 31 × for the multi-device multi-realization approach, as shown in Fig. 9(b). The observed differences in performance are related not only with the GPU device intrinsic characteristics (memory bandwidth, occasional warp-divergence and infrequent non-coalesced memory accesses), but also with dataset related parameters, such as the simulation grid size (N), the amount of experimental data available (N_r), the number of classes of the bi-distribution function used during the co-simulation procedure (N_c), and simulation specific parameters, such as the number neighbouring data used to build the kriging system (k), which was herein kept constant with a value of 12. Accordingly, the time complexity of the algorithm implementation is given by:

$$O\left((N-N_r)\left(\frac{(N-N_r)-1}{2}+k^2+N_c\log(N_c)\right)\right)$$

In what concerns the device memory occupation, it was verified that the memory complexity of the proposed implementation is linearly dependent with the simulation grid size (N) and with the

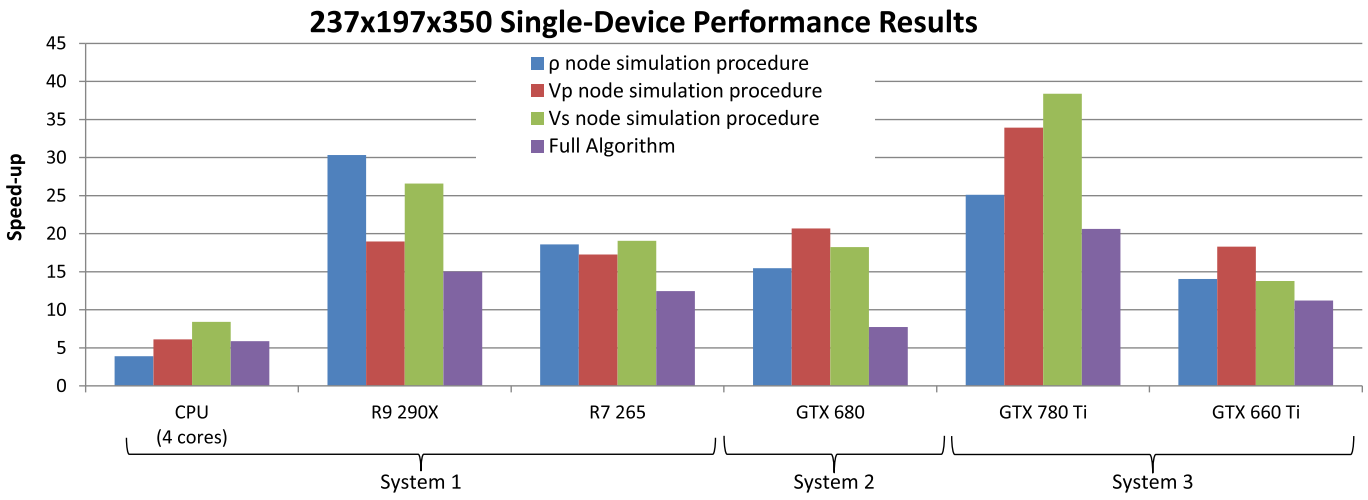

Fig. 8. Single-device execution speed-up results considering the dataset with $237 \times 197 \times 350$ nodes.

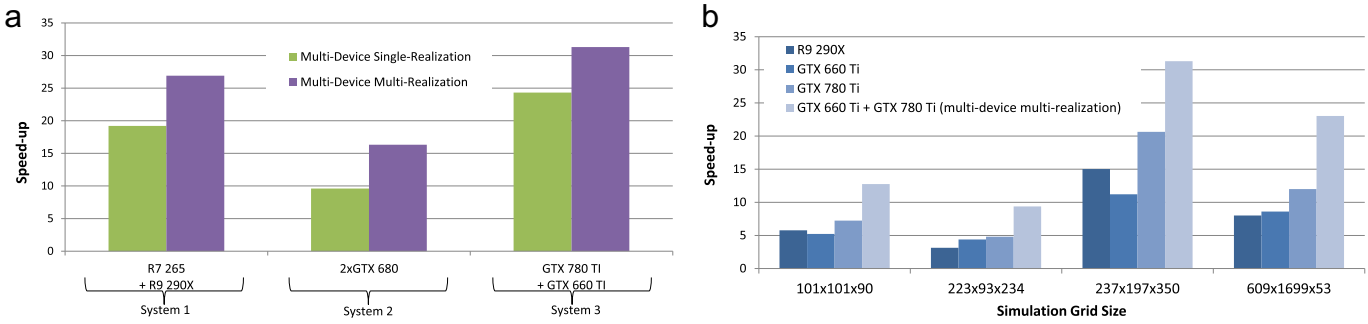

Fig. 9. Execution speed-up results compared to the sequential CPU implementation (executed in system 1). (a) Multi-device results for the dataset with $237 \times 197 \times 350$ nodes and (b) obtained performance results when processing multiple datasets.

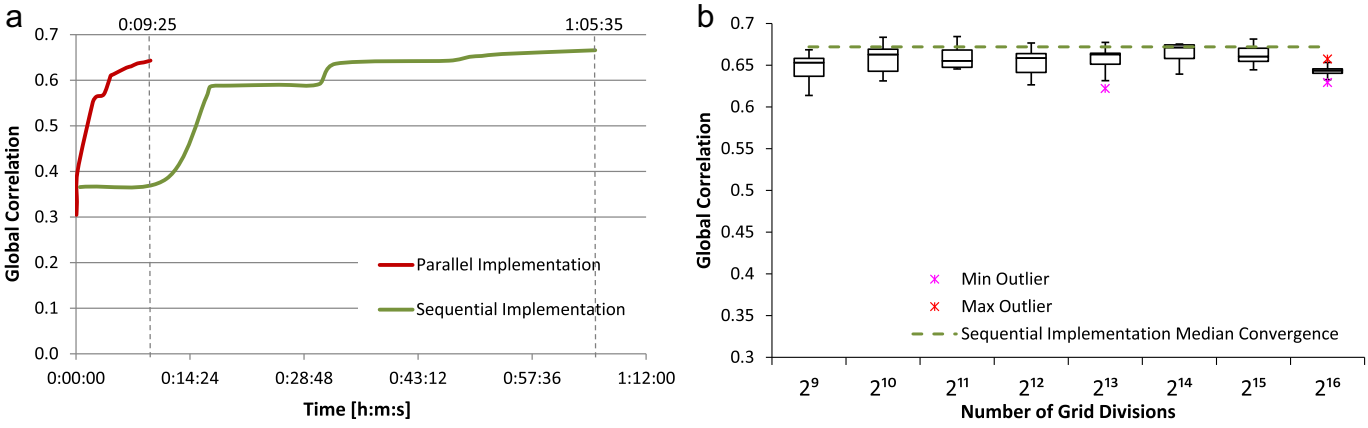

Fig. 10. Graphical analysis of the obtained results comparing the parallel with the sequential implementations of the algorithm, composed of 6 iterations each with 32 sets of simulations. The dataset size is $101 \times 101 \times 90$ nodes. (a) Convergence analysis over time and (b) convergence analysis when varying the number of grid divisions.

number of classes (N_c) of the bi-distribution function used during the co-simulation procedure: $O(N + N_c)$. Accordingly, in a worst case perspective (when all layers are simultaneously allocated in the GPU) the multi-device multi-realization implementation required 1080 MB of device memory for the largest dataset (composed of approximately 58 million nodes), in contrast with the 510 MB required for the dataset with $237 \times 197 \times 350$ nodes.

5.2. Quality of the results

The quality of the retrieved inversion models regarding the attained convergence is depicted in Fig. 10. The assumptions related with the spatial continuity pattern and prior distributions were kept constant for both cases. In particular, by analysing the chart in Fig. 10(a), it can be verified that despite the significantly faster convergence rate of the parallel implementation, similar global correlation coefficients (computed by comparing the synthetic with the real seismic models) were obtained for 6 iterations (each composed of 32 sets of simulations) both in the parallel and in the sequential implementations. In fact, as it can be observed in Fig. 10(b), where 10 independent runs are considered for each number of grid divisions (which corresponds to the nodes being simulated in parallel), satisfactory results in terms of convergence were obtained even when large amounts of grid divisions were

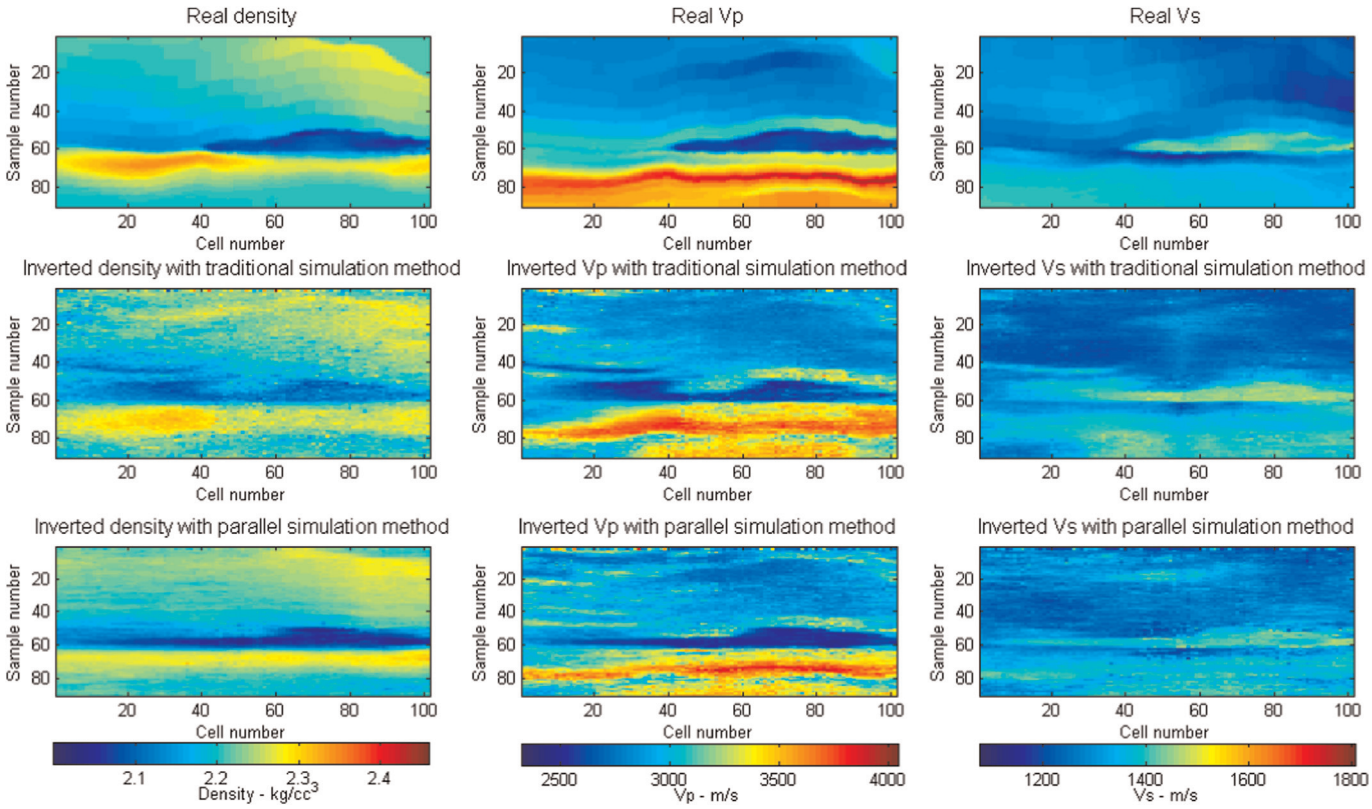

Fig. 11. Comparison of the spatial distribution of the simulated physical properties in a given realization of the inversion procedure, using the $101 \times 101 \times 90$ dataset.

considered, demonstrating that the adopted relaxation aiming an efficient parallelization do not significantly affect the quality of the results. This can be justified with the considered postponing optimization that avoids the simulation of nodes that have few or no conditioning data in the neighbour blocks. As a result, when the block size becomes smaller (as a result of increasing the number of grid divisions), a consistent spatial distribution of the physical property being simulated is granted during the first steps of the simulation, thus ensuring the algorithm convergence.

Fig. 11 illustrates the mean model computed with all the models generated during the last iteration for the proposed parallelization approach, together with the corresponding sequential approach, for the considered physical properties (ρ, Vp and Vs). By the interpretation of these vertical sections, it can be observed that the original spatial distributions of the physical properties being simulated are guaranteed, despite the relaxation that was introduced to improve the parallelization. In addition, both approaches are able to reproduce the main features as observed in the real elastic models. As in the sequential approach, the parallelized methodology is able to better reproduce Vp and density while struggles to reproduce correctly Vs. This effect is a consequence of the cascade approach for generating the set of elastic models. Nevertheless, it should be recalled that the inverted and the real models are not expected to be exactly the same, given the stochastic nature of the DSS algorithm.

6. Conclusions

Accurate characterization of oil and gas reservoirs is a fundamental procedure to support the decision making process in oil and gas industry. However, improving the accuracy of the earth subsurface models often results in long (sometimes unfeasible) computational times. To circumvent such an issue, this paper proposes novel algorithm parallelization approaches that allow to efficiently take advantage of GPU-based systems. This is achieved by adopting a careful and selective spatial relaxation of the simulation grid dependencies, such as to increase the available parallelism opportunities and to improve the utilization of the GPU computational resources.

To efficiently exploit multi-GPU heterogeneous systems, two multi-device parallelization approaches were proposed. One of the approaches exploits fine-grain *path-level* parallelism, by strictly distributing the nodes of the simulation grid to the available devices, according to their computational capabilities. The other approach exploits *realization-level* parallelism, in order to efficiently distribute the parallel computation across the available processing devices. Although the two approaches are not mutually exclusive, the second approach allows a better distribution of the workload and helps minimizing the computation overheads. Accordingly, a significant reduction of the execution time is achieved by effectively exploiting the computational capabilities (leading to speed-ups as high as $31\times$ in a dual-GPU system) without compromising the quality of the obtained inversion results.

Acknowledgements

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project THREadS (ref. PTDC/EEA-ELC/117329/2010) and project UID/CEC/50021/2013.

References

Azevedo, L., Nunes, R., Soares, A., Neto, G., 2013. Stochastic seismic AVO inversion. In: 75th EAGE Conference & Exhibition, pp. 10–13.
Azevedo, L., Nunes, R., Soares, A., Neto, G., 2014. Geostatistical seismic AVO inversion: real case application. In: SEG Annual Meeting.

Bosch, M., Mukerji, T., Gonzalez, E.F., 2010. Seismic inversion for reservoir properties combining statistical rock physics and geostatistics: a review. Geophysics 75 75A165–75A176.

Buland, A., Omre, H., 2003. Bayesian linearized AVO inversion. Geophysics 68, 185–198.

Doyen, P., 2007. Seismic reservoir characterization: an earth modelling perspective. In: European Association of Geoscientists & Engineers (EAGE).

Ferreirinha, Tomás and Nunes, Rúben and Soares, Amílcar and Pratas, Frederico and Tomás, Pedro and Roma, Nuno}, pages={239-250, 2014, isbn={978-3-319-14324-8, booktitle=Euro-Par 2014: Parallel Processing Workshops, volume=8805, series=Lecture Notes in Computer Scienc}, editor=Lopes, Luís and Žilinskas, Julius and Costan, Alexandru and Cascella, RobertoG. and Kecskemeti, Gabor and Jeannot, Emmanuel and Cannataro, Mario and Ricci, Laura and Benkner, Siegfried and Petit, Salvador and Scarano, Vittorio and Gracia, José and Hunold, Sascha and Scott, StephenL. and Lankes, Stefan and Lengauer, Christian and Carretero, Jesus and Breitbart, Jens and Alexander, Michael, doi=http://dx.doi.org/10.1007/978-3-319-14325-5_21, title=GPU Accelerated Stochastic Inversion of Deep Water Seismic Data, url=http://dx.doi.org/10.1007/978-3-319-14325-5_21, publisher=Springer International Publishing.

Horta, A., Soares, A., 2010. Direct sequential co-simulation with joint probability distributions. Math. Geosci. 42, 269–292.

Hysing, A.D., 2010. Parallel Seismic Inversion for Shared Memory Systems (Ph.D. thesis). Norwegian University of Science and Technology.

Ilic, A., Pratas, F., Sousa, L., 2014. Cache-aware Roofline model: Upgrading the loft. Comput. Archit. Lett. 13 (1), 21–24. http://dx.doi.org/10.1109/L-CA.2013.6, ISSN: 1556-6056.

Mallick, S., 1995. Model-based inversion of amplitude-variations-with-offset data using a genetic algorithm. Geophysics 60, 939–954.

Mariethoz, G., 2010. A general parallelization strategy for random path based geostatistical simulation methods. Comput. Geosci. 36, 953–958.

Nunes, R., Almeida, J.A., 2010. Parallelization of sequential Gaussian, indicator and direct simulation algorithms. Comput. Geosci. 36, 1042–1052.

Shuey, R., 1985. A simplification of the Zoeppritz equations. Geophysics 50, 609–614.

Soares, A., 2001. Direct sequential simulation and cosimulation. Math. Geol. 33, 911–926.

Stinessen, B.O., 2011. Profiling, Optimization and Parallelization of a Seismic Inversion Code (Ph.D. thesis). Norwegian University of Science and Technology.

Tahmasebi, P., Sahimi, M., Mariethoz, G., Hezarkhani, A., 2012. Accelerating geostatistical simulations using graphics processing units (GPU). Comput. Geosci. 46, 51–59.

Taniça, L., Ilic, A., Tomás, P., Sousa, L., 2014. Schedmon: a performance and energy monitoring tool for modern multi-cores. In: 7th International Workshop on Multi/many-Core Computing Systems.

Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. Society for Industrial and Applied Mathematics (SIAM).

Vargas, H., Caetano, H., Filipe, M., 2007. Parallelization of sequential simulation procedures. In: EAGE Petroleum Geostatistics.