Accelerating a Stochastic Seismic Inversion Algorithm using OpenCL-based Heterogeneous Platforms

Tomás Novais Ferreirinha

Abstract—Seismic inversion algorithms have been playing a key role in the characterization of oil and gas reservoirs, where the retrieved subsurface elastic models need to be reliable to improve decision making, for example decisions about well locations, thus maximizing the extraction of such resources. Since these algorithms usually rely in computer simulations that generate, process and store huge amounts of data, their usage in the industry is often limited by their long execution times. As a result, the demand for an efficient and accelerated execution of these algorithms is not only required to decrease the time to decision, but also to allow the development of larger and higher resolution models of the subsurface. In accordance, this thesis proposes a novel parallelization approach of a state of the art Stochastic Seismic Amplitude versus Offset Inversion algorithm, exploiting heterogeneous computing platforms based on a unified OpenCL programming framework, thus enabling the application to be executed in devices with different architectures and from different vendors. To take full advantage of the computational power made available by systems composed of multiple (and possibly different) CPUs and GPUs, a spatial division of the simulation space is performed, enabling the parallel simulation of multiple regions of the geological model. With this approach, a performance speed-up as high as 27.65× using two distinct GPUs was achieved, without compromising the accuracy of the obtained models.

Index Terms—Heterogeneous computing, graphics processing unit, OpenCL, seismic inversion, parallel computing.

I. 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. As a result of the increased computing capabilities that have become available to this prospecting industry, complex computational models of the Earth’s subsurface can now be applied to locate and estimate reserves and to diagnose and improve the performance of the producing fields. 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 in most cases.

Stochastic algorithms have been playing 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 [1], based on a Direct Sequential Simulation (DSS) [2] 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 the best of the authors knowledge, only one parallel implementation of the DSS algorithm was proposed to exploit multi-core Central Processing Units (CPUs) [3], by applying a straightforward functional decomposition of the algorithm. Such approach presented considerable limitations not only in terms of scalability but also in the capacity to exploit modern heterogeneous computing systems composed of multiple processors. Nevertheless, the generic parallelization of geostatistical simulation methods has already been studied [4], where three distinct 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 [3]–[6] and Graphics Processing Units (GPUs) [7]. 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 herein proposed, by considering highly heterogeneous platforms composed by 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. 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
a 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 II. The proposed parallelization approach is described in detail in Section III, serving as a background to the following section where the single and multi-device parallel solutions are described. An experimental evaluation of the developed implementations is presented in Section V. Finally, Section VI concludes this paper, summarizing the main conclusions and contributions.

II. STOCHASTIC SEISMIC AVO INVERSION

Seismic reflection data plays a key role in hydrocarbon reservoir geo-modelling work-flows. 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 [8]. 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 [9]. Unfortunately, the integration of the seismic reflection data is not straightforward within the modelling work-flow, 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 [9]. However, due to measurement errors and approximations in the physical models that are often used to solve the seismic inverse problem, this is a ill-posed and highly non-linear problem with a non-unique solution [9], [10].

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., [11], [12]). 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 the art geostatistical pre-stack seismic inversion methodology [1], [13] that allows the inversion of pre-stack seismic data directly for density, P-wave and S-wave velocity 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 on an iterative geostatistical inverse procedure based on two main concepts: 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 [2] and co-DSS with joint probabilities distribution [14]). 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 Figure 1):

- Joint simulation of density, P-wave and S-wave velocities
  1) Stochastic simulation of $N_s$ density models conditioned to the available well-log data with DSS [2];
  2) Stochastic co-simulation of $N_s$ P-wave velocity models given the $N_s$ previously simulated density models with co-DSS with joint-distributions [14];
  3) Stochastic co-simulation of $N_s$ S-wave velocity models given the $N_s$ previously simulated P-wave velocity models with co-DSS with joint-distributions [14];
- Optimization for the iterative convergence
  4) From the triplet of elastic models computed in the previous stage, calculation of $N_s$ angle-dependent synthetic pre-stack seismic volumes using Shueys linear approximation [15];
  5) Comparison between each synthetic angle gather with the corresponding real angle gather on a trace-by-trace basis; and calculation of $N_s$ local correlation gathers for each location within the seismic grid;
  6) Selection of the areas with the highest correlation coefficient, by simultaneously considering all angles to build the best $ρ$, Vp and Vs models, by using a genetic based algorithm;
  7) Compute the local correlation model regarding the best generated, Vp and Vs models; Iterate from 1) until the matching criteria (i.e., global correlation between the original and the synthetic pre-stack seismic) is reached.

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

A. Randomly selection of a node from a regular grid;
B. Construction and solution of a kriging system for the selected node, by linear interpolating the previously selected conditioning data [16];
C. Estimation of a Local Cumulative Distribution Function (LCDF) at the selected node, by using the previously computed simple kriging estimate that considers both the original and the previously simulated data [2];
D. Draw 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.

The co-simulation variant of this algorithm enables the simulated variable to be conditioned by other previously simulated variables, without any prior transformation of the secondary variable. 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). 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 [8].

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.

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 regards to recorded pre-stack seismic data. In addition, the corresponding density, P-wave and S-wave velocity models can also be retrieved. This methodology was already successfully tested and implemented in a synthetic seismic dataset [1] and it is herein applied to the same challenging synthetic deep-offshore reservoir.

III. ALGORITHM PARALLELIZATION

In order to efficiently parallelize an application, it is important to define how performance is going to be measured and to identify where and how the parallelization effort is going to be applied. For such purpose, an extensive analysis of the algorithm processing modules was conducted and is herein briefly described. Such analysis supports a good decision in terms of a partitioning of the algorithm, which is essential in order to enable the development of efficient and highly scalable implementations.

A. 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 using the Performance Application Programming Interface (PAPI) [17] to interface with the hardware performance counters, 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 101x101x90 nodes; and a larger and more realistic one, with 237x197x350 nodes. From the obtained profiling results, it has been observed that more than 95% of the algorithm execution time is spent in the generation of Np/Vp models using the DSS algorithm (see blocks 1, 2 and 3 in Figure 1), most specifically during the node simulation procedure. Consequently, this part of the algorithm was chosen as the prime focus for acceleration by taking advantage of heterogeneous computing, since it is the part where the parallelization effort may result in a higher speed-up. 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 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 part 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 Figure 3). This makes the data dependencies difficult to predict or to confine to a well defined spatial region, since it initially depends on the distribution of well-log data and subsequently depends on the positions of the nodes being simulated. 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.
B. 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 Figure 4), 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. As a matter of fact, by only accelerating parts A and B, a maximum speed-up of $1.4 \times$ is achievable. Furthermore, an individual acceleration of each part of the algorithm would hardly give good results too, 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 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 Figure 5). Different number of divisions result in different generated models, because the random-paths are going to be different, and consequently nodes are going to be simulated under different circumstances (being conditioned by different data). However, this number of divisions must be great enough in order to completely exploit the computing capabilities of the considered devices. At this respect, it is worth noting that the considered sub-divisions of the simulation grid are not required to be cubic. In fact, the anisotropic nature of the well-log data being processed (sparsely located along the study area but with a very high vertical resolution) suggests that there are significantly less dependencies in the vertical direction, which allows for a greater vertical division of the simulation grid.

Since this approach implies the removal of some data dependencies, the choice of the nodes to be simulated in parallel must be performed with some criteria, in order to maximize the performance of the algorithm execution and still ensure the reproduction of the spatial distributions and inherent uncertainties of the physical properties being simulated ($V_p$ and $V_s$). Accordingly, 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 [18], 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 have been observed, the second alternative was chosen for further evaluation (see Section V), since providing a greater flexibility to the random path generation is usually a major concern in real-life case studies.

IV. IMPLEMENTATION IN GPU-BASED HETEROGENEOUS PLATFORMS

Taking into account the presented parallelization approach, the devised methodology to efficiently exploit heterogeneous platforms composed both by CPUs and GPUs are herein described. For such purpose, a brief introduction to GPU programming is presented, together with a detailed description of a single-device and two distinct multi-device implementations.

A. 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). Therefore, whenever different threads of the same warp need to execute different instructions (usually a consequence of if(condition) statements), there is a significant loss of parallel efficiency, since all threads in the warp will require to pass through the instructions of both conditional branches. 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.

B. Single-Device Parallel Solution

As presented in Section III-B, the proposed parallelization approach considers a spatial division of the simulation grid (used by the DSS algorithm), in order to allow the parallel simulation of multiple nodes. According to the OpenCL programming model, a data parallel execution of multiple node simulations is performed by having several instances of the same kernels being executed at the same time by multiple work-groups. However, since GPUs are being considered as target devices, those kernels have to be carefully crafted in order to efficiently exploit the GPU architecture, by minimizing performance loss factors (warp-divergence, non-coalesced memory accesses) and maximizing the device occupancy. Therefore, when migrating the node simulation procedure of the DSS algorithm into OpenCL code, the application has to be functionally partitioned into multiple kernels, achieving a compromise between the granularity of the kernels (influencing the number of registers being used and consequently the occupation of the GPU) and the number of kernel calls (that have an inherent overhead). After implementing and optimizing the proposed parallelization approach, the resulting implementation considered 4 distinct kernels with the following functionalities:

i) Selection of conditioning data for the nodes being simulated in parallel (one from each sub-grid);
ii) Construction of the kriging systems, according to the locations of the selected conditioning data and subsequently solve them and estimate the corresponding lcdfs;
iii) Condition the distribution functions by a previously simulated auxiliary variable and generation of the simulated values given the resulting lcdfs by using a Monte Carlo method;
iv) Store the simulated values into the simulation grid and update the information relative to the already simulated nodes (mean and variance).

It must be noted that, with the exception of a task that will be further detailed in Section IV-E3, only tasks related to the node simulation procedure are actually performed by the OpenCL enabled 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 Figure 6). The acceleration of other parts of the algorithm was not considered because they did not represent a significant part of the algorithm execution time.

1) Mapping Methodology: 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. The main difference regarding both approaches is concerned with the number of nodes being simulated in parallel, 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, 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 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.

C. Multi-Device Single-Realization Parallel Solution

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. Therefore, at each step of the simulation procedure the sub-grids are re-distributed according to a traditional iterative load balancing routine [19].

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 Figure 7). 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.

D. Multi-Device Multi-Realization Parallel Solution

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 set of simulations ($\rho$, $V_p$, $V_s$), together with the corresponding forward model and the correlation related computations regarding those sets of simulations (see Figure 8).

Nevertheless, the construction of the best model still needs to 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 set of CPU threads 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 Figure 8).

To maximize the performance, all devices iteratively simulate one realization without any midpoint synchronization. However, when a set of $N_s$ 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 Figure 8), 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.

E. Considered Optimizations

Several optimizations were considered in order to optimize the performance of the proposed implementations. Such optimizations arise both from good GPU programming practices, and from a continuous profiling of the implemented solutions, in order to assess the main limitations and overheads. For such purpose, those optimizations were divided into three different categories: generic optimizations, which are mainly related with good GPU programming practices that can be similarly applied in other GPU programming projects; application specific optimizations, which are related with limitations intrinsic to the algorithm characteristics; and optimizations to the rest of the algorithm, which do not specifically concern the part being accelerated by using GPUs.

1) Optimization of the DSS Algorithm Kernels: To envisage a high performance implementation of the DSS algorithm, several optimization procedures are worth to be considered. 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. Furthermore, the inherent parallelization overhead was minimized by performing the data transfers with a minimum number of OpenCL enqueue buffer calls as possible. To attain this objective, a compromise was achieved between the granularity of the kernels (influencing 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.

2) Application Specific Optimizations: 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 scariness 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, 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 V-B).

3) Other Optimizations: Some optimizations outside the main simulation procedure were also considered in the presented implementation. 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). Moreover, the output files that have to be written during the execution of the algorithm are also written in parallel by a different CPU thread, in order to avoid as much overhead as possible, thus becoming completely overlapped with the GPU computations. The acceleration of other parts of the algorithm was also considered. However, the characteristics of such parts were not appropriate for GPU computing (e.g., the generation of the random path presented to be inefficient, mainly due to the warp divergence introduced by the simulation postponing optimization and by the non-coalesced memory accesses caused by its random behaviour).

V. 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 both cases, the execution was evaluated using two distinct datasets: one with 101x101x90 nodes and another (more realistic) with 237x197x350 nodes.

A. Performance Improvements

In order to measure the performance of the proposed parallelization approaches, the execution times using multiple heterogeneous environments (see Table I) 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 5 iterations composed by 8 sets of simulations each over the larger dataset.

Figure 9(a) 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 18.06×, when considering the execution of the algorithm using a GTX 780 Ti GPU, and a performance improvement of 5.90× when the algorithm was mapped onto 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.

When analysing the obtained results from the perspective of the multi-device approaches (see Figure 9(b)), it can be observed that significantly higher throughputs were effectively attained when considering the multi-device multi-realization approach. In particular, a speed-up of 27.65× was obtained when executing the algorithm over both the GPUs from System 3. Such already expected acceleration rate arises not
only because a greater part of the algorithm is actually being executed in parallel, but also because there is no communication and synchronization overheads during the node simulation procedure. In what concerns the multi-device single-realization approach, although the obtained speed-up is slightly below the theoretical limit when considering a simplistic perspective that simply counts the number of used devices (a speed-up of \(2\times\) is theoretically expected when using two similar accelerators, compared to the use of only one), the load was successfully divided between the available devices, being the difference to the theoretical limit mainly justified by the sequentially computed parts of the algorithm and by the communication and parallelization overheads.

B. Quality of the Results

The quality of the retrieved inversion models regarding the attained convergence is depicted in Figure 10. The assumptions related with the spatial continuity pattern and prior distributions was kept constant for both cases. In particular, by analysing the chart in Figure 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 by 32 sets of simulations) both in the parallel and in the sequential implementations. In fact, as it can be observed in Figure 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 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.

Figure 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 (\(\rho\), 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.

VI. CONCLUSIONS

An efficient parallelization of a state of the art stochastic seismic AVO inversion algorithm in heterogeneous platforms was proposed in this paper. The acceleration of such algorithms not only allows for faster reservoir modelling, but also greatly facilitates and provides the feasible means 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 an appropriate compensation of the adopted division of the simulation grid, thus allowing the parallel simulation of multiple nodes corresponding to different regions of the model. According to the obtained experimental results, the proposed approach not only significantly reduces the algorithm execution time (achieving multi-device speed-ups of over \(27\times\)), but also still guarantees the quality of the obtained inversion results.

REFERENCES

Global Correlation

Time \[h:m:s\]

Parallel Implementation

Sequential Implementation

0:08:46
1:25:25

(a) Convergence analysis over time.

(b) Convergence analysis when varying the number of grid divisions.

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 101x101x90 nodes.

Fig. 11. Comparison of the spatial distribution of the simulated physical properties in a given realization of the inversion procedure, using the 101x101x90 dataset.


