Towards seismic wave modeling on heterogeneous many-core architectures using task-based runtime system

Víctor Martínez, David Michéa, Fabrice Dupros, Olivier Aumage, Samuel Thibault, Hideo Aochi, Philippe Olivier Alexandre Navaux

Towards seismic wave modeling on heterogeneous many-core architectures using task-based runtime system

Víctor Martínez*, David Michéa‡, Fabrice Dupros‡, Olivier Aumage§,
Samuel Thibault§, Hideo Aochi‡ and Philippe O. A. Navaux*
* Informatics Institute, Federal University of Rio Grande do Sul (UFRGS),
Av. Bento Gonçalves, 9500, Campus do Vale, 91501-970, Porto Alegre, Brazil
{victor.martinez,navaux}@inf.ufrgs.br
‡ BRGM, 3 Av. Claude Guillemin, Orléans, France
{d.michea,f.dupros,h.aochi}@brgm.fr
§ Inria Bordeaux Sud-Ouest, Bordeaux, France
{olivier.aumage,samuel.thibault}@inria.fr

***Abstract*—Understanding three-dimensional seismic wave propagation in complex media remains one of the main challenges of quantitative seismology. Because of its simplicity and numerical efficiency, the finite-differences method is one of the standard techniques implemented to consider the elastodynamics equation. Additionally, this class of modelling heavily relies on parallel architectures in order to tackle large scale geometries including a detailed description of the physics. Last decade, significant efforts have been devoted towards efficient implementation of the finite-differences methods on emerging architectures. These contributions have demonstrated their efficiency leading to robust industrial applications. The growing representation of heterogeneous architectures combining general purpose multicore platforms and accelerators leads to re-design current parallel application. In this paper, we consider StarPU task-based runtime system in order to harness the power of heterogeneous CPU+GPU computing nodes. We detail our implementation and compare the performance obtained with the classical CPU or GPU only versions. Preliminary results demonstrate significant speedups in comparison with the best implementation suitable for homogeneous cores.**

I. Introduction

The trend at the hardware level is to increase the complexity of available computing node. This includes several level of hierarchical memories, increasing number of heterogeneous cores or low-level optimization mechanisms. Another concern is coming from the increasing gap between the computing power and the cost for data transfers. These evolutions lead to progressively re-design current applications that mainly exploit flat programming models. The challenge is therefore to decouple as much as possible the algorithms and the knowledge of the underlying architecture.

Consequently, many runtime systems have been designed for programming and running applications on heterogeneous architectures with accelerators; examples include *G-Charm* [1], a framework for execution of message-driven parallel applications on hybrid systems, based on Charm++, UTK's PaRSEC [2] framework, as well as OmpSs [3] from the Barcelona Supercomputing Center. In [4] the *XKaapi* for data-flow task programming model on heterogeneous architectures is presented. *StarPU* is a tasking API that provides numerical kernel designers with a convenient way to execute parallel tasks over heterogeneous hardware on the one hand, and easily develop and tune scheduling algorithms on the other hand. StarPU is based on the integration of the data-management facility with a task execution engine [5].

In this paper, we focus on seismic wave propagation algorithm that is routinely both in the oil and gas industry and in strong motion analysis in seismology. The finite-differences numerical method used for this problem also lies at the heart of a significant fraction of numerical solvers in other fields. In terms of computational efficiency, one of the main difficulties is to deal with the disadvantageous ratio between the limited pointwise computation and the intensive memory access required, leading to a memory-bound situation. This situation is rather challenging for heterogeneous platforms as one of the main difficulty is to deal with the costly memory transfers.

We therefore introduce an optimized implementation of seismic wave propagation model suitable for heterogeneous architectures and we demonstrate the efficiency of our approach on two heterogeneous architectures using the maximum number of processing units available. The paper proceeds as follows. Section II discusses the fundamentals of seismic wave simulation and presents the application under study (i.e. *Ondes3D*). Classical implementations of finite-differences method, both on multicore and GPU architectures are described at this level. Section III introduces StarPU runtime system. Section IV describes the task-based implementation of Ondes3D software package on top of StarPU. Section V and VI discusses the performances obtained with the influence of additional parameters of the task-based parallel algorithm. Section VII describes related work and Section VIII concludes this paper.

II. Seismic wave propagation

In this section we introduce the governing equations associated with the three-dimensional modelling of seismic wave propagation in elastic media. We describe the standard implementations of the finite-differences discretization for x86 cores and for GPU platforms corresponding to Ondes3D application developed by the French Geological Survey (BRGM).

A. Elastodynamic equation

Seismic waves radiated from an earthquake are often simulated under the assumption of an elastic medium although the waves attenuate due to some anelasticity. Let us consider a 3D isotropic elastic medium, the seismic wave equation is given by :

$$\begin{cases} \rho \frac{\partial}{\partial t} v_x &= \frac{\partial}{\partial x}\sigma_{xx} + \frac{\partial}{\partial y}\sigma_{xy} + \frac{\partial}{\partial z}\sigma_{xz} + f_x \\ \rho \frac{\partial}{\partial t} v_y &= \frac{\partial}{\partial x}\sigma_{yx} + \frac{\partial}{\partial y}\sigma_{yy} + \frac{\partial}{\partial z}\sigma_{yz} + f_y \\ \rho \frac{\partial}{\partial t} v_z &= \frac{\partial}{\partial x}\sigma_{zx} + \frac{\partial}{\partial y}\sigma_{zy} + \frac{\partial}{\partial z}\sigma_{zz} + f_z \end{cases} \quad (1)$$

$$\begin{cases} \frac{\partial}{\partial t}\sigma_{xx} &= \lambda \left(\frac{\partial}{\partial x} v_x + \frac{\partial}{\partial y} v_y + \frac{\partial}{\partial z} v_z \right) + 2\mu \frac{\partial}{\partial x} v_x \\ \frac{\partial}{\partial t}\sigma_{yy} &= \lambda \left(\frac{\partial}{\partial x} v_x + \frac{\partial}{\partial y} v_y + \frac{\partial}{\partial z} v_z \right) + 2\mu \frac{\partial}{\partial y} v_y \\ \frac{\partial}{\partial t}\sigma_{zz} &= \lambda \left(\frac{\partial}{\partial x} v_x + \frac{\partial}{\partial y} v_y + \frac{\partial}{\partial z} v_z \right) + 2\mu \frac{\partial}{\partial z} v_z \\ \frac{\partial}{\partial t}\sigma_{xy} &= \mu \left(\frac{\partial}{\partial y} v_x + \frac{\partial}{\partial x} v_y \right) \\ \frac{\partial}{\partial t}\sigma_{xz} &= \mu \left(\frac{\partial}{\partial z} v_x + \frac{\partial}{\partial x} v_z \right) \\ \frac{\partial}{\partial t}\sigma_{yz} &= \mu \left(\frac{\partial}{\partial z} v_y + \frac{\partial}{\partial y} v_z \right). \end{cases} \quad (2)$$

In the previous equations, v and σ represent the velocity and the stress field respectively and f denotes a known external source force. The medium is characterized by the elastic (Lamé) parameters λ and μ and ρ is the density.

B. Numerical method and standard implementations

Due to its simplicity, the finite-differences method is widely used to compute the propagation of seismic waves. The numerical kernel under study relies on the classical 4-th order in space and second-order in time approximation [6], [7].
Figure II.1 provides an overview of the computational flowchart where indices i, j, ij represent a component of a vector or tensor field in Cartesian coordinates (x, y, z), v_i and σ_{ij} represent the velocity and stress field. After a pre-processing phase devoted to the initialization of the velocity, stress and external force components, the algorithm goes through a time-dependent double triple-nested loops. The stress and velocity components are evaluated following and odd-even dependency (*i.e.*the computation of the stress field reuses the results of the update of the velocity field).

Algorithm II.1: FINITE-DIFFERENCES FLOWCHART(σ, v)

for $x \leftarrow 1$ **to** $x\_dimension$
do { **for** $y \leftarrow 1$ **to** $y\_dimension$
do { **for** $z \leftarrow 1$ **to** $z\_dimension$
do {COMPUTE_VELOCITY(σ_{ij})
for $x \leftarrow 1$ **to** $x\_dimension$
do { **for** $y \leftarrow 1$ **to** $y\_dimension$
do { **for** $z \leftarrow 1$ **to** $z\_dimension$
do {COMPUTE_STRESS(v_i)

C. Reference parallel implementations

1) CPU: On shared-memory architectures, a popular way to extract the parallelism for such applications is to exploit the triple nested loops coming from the spatial dimensions of the problem under study. This strategy allows a straightforward benefits of OpenMP directives. Additional optimizations should be considered in order to limit the impact of NUMA (Non-Uniform Memory Access) architectures. The kernel used in this paper is similar to the implementation described in [8].

2) GPU: Regarding the finite-differences method, several applications have been ported to GPUs, particularly for seismic wave modelling [9]–[11]. The high reading redundancy (13/1) coming from the fourth-order stencil used in space makes GPU a very efficient architecture for such memory-bound applications. A full description of the implementation corresponding to Ondes3D application could be found in [12].

III. STARPU RUNTIME SYSTEM

The StarPU [5] Runtime System developed by Team STORM provides a framework for task scheduling on heterogeneous platforms. It is able to calibrate the relative performance of multiple, heterogeneous implementations of computing kernels, as well as the cost of data transfers between the main memory space and accelerator memory spaces, such as to optimize work mapping dynamically among heterogeneous computing units, during the execution of the application. This scheduling framework jointly works with a distributed shared-memory manager in order to optimize data transfers, to perform replication and consistency management for avoiding redundant transfers, and to overlap communications with computations. A complete description of the available schedulers could be found in [13]. In this study, we consider the following scheduling algorithms :

eager	A central queue from which all workers pick tasks concurrently.
ws	A work-stealing scheduler, where idle workers may steal work from busy workers.
dm	The Deque Model (DM) scheduler maps tasks onto workers using an history-based kernel performance model.
dmda	An variant of the DM scheduler also taking transfer costs into account.
dmdar	A variant of the DMDA scheduler such that per-worker task queues are sorted according to the number of already available dependent pieces of data.

IV. ELASTODYNAMICS NUMERICAL KERNEL ON TOP OF STARPU RUNTIME SYSTEM

Finite-differences methods naturally express data parallelism whereas StarPU works at tasks level. In order to express task-based parallelism with such numerical approach, one needs to split the model into a fine-grained grid of blocks where each part of the code (mainly the update of the velocity and the stress components) is executed on a single block as a computing task. Each block includes inner grid-points corresponding to the physical domain and outer ghosts zones

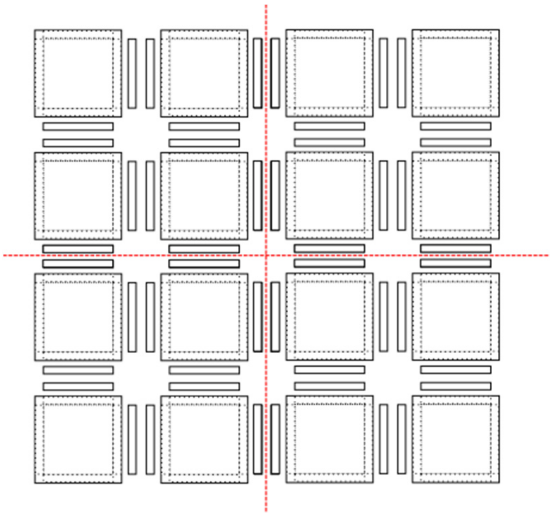


Figure 1: Grid of blocks including inner grid-points corresponding to the physical domain and outer ghosts zones

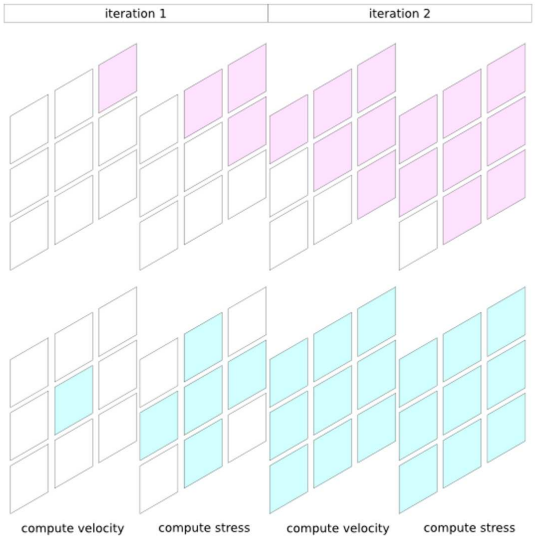


Figure 2: Tasks dependency on a grid of 3×3 blocks

for the grid points exchanged between neighboring subdomains 1. The three-dimensional domain is cutted along the horizontal directions as models in seismology often describe a thin and wide crust plate in order to evaluate surface effects. A naive implementation would require expensive copies between blocks because the boundary data are not contiguous in memory (several calls of *memcpy* (or *cudaMemCpy*) for small size data). Therefore a GRAM buffer which is filled using a CUDA kernel is created and then copied only once. StarPU framework allows us to simply define handlers bound to the data in order to smoothly manage the mapping on the heterogeneous architecture.

The task-based parallelism model leads to the creation of a large number of tasks, ideally loosely coupled. StarPU framework arranges these tasks in a Directed Acyclic Graph (DAG) according to the data dependency. Typically, we can identify the tasks devoted to data transfers (used to save data from block boundaries to buffers and vice versa) and the two computing kernels (computation of the six stress and the three velocity components). For each timestep, the same pattern of task creation is repeated. At this stage, we have simplified the original implementation by discarding the absorbing boundary conditions. As these numerical conditions introduce load imbalance at the boundaries of the computational domain, it makes much more complex the analysis of the results on heterogeneous platforms [8]. The time spent in data management kernels is very small compared to the elapsed time for the computation kernels. Nevertheless, these tasks are crucial as they express the dependencies between blocks that are adjacent in horizontal directions.

Figure 2 illustrates this situation considering a grid of 3×3 blocks. For instance, if a single task is scheduled on a slow computing resources and the remaining tasks are executed on faster resources, the colored tasks cannot be scheduled before finishing the the first one. The main programming effort has been the creation of the relevant CPU and the GPU kernels corresponding to the numerical scheme. Figure 3 provides a detailed description of the algorithm.

```
1:  StarPU Initialization
2:  procedure Initialization of data structures
3:      Read_parameters(file)
4:      Model_slicing(Create blocks)
5:      Allocate_memory(blocks, nodes)
6:      Read(sources, source time, stations positions)
7:      Read(model)
8:      Set(material properties)
9:  end procedure
10: procedure Create all tasks
11:     Create dummy start task(Time measurement)
12:     for each iteration do
13:         for each block do
14:             Create task(update source)
15:             Create task(compute velocity)
16:             Create task(save boundaries for velocity)
17:             Create task(compute stress)
18:             Create task(save boundaries for stress)
19:             Create task(record seismograms)
20:         end for
21:         Create dummy end task(Synchronization)
22:     end for
23:     Release dummy start task
24:     Wait(end of the dummy end task)
25: end procedure
```

Figure 3: Task-based workflow for *Ondes3D* application

V. Experimental setup

In this section we describe the architectures used for our experiments.

1) **Commodity Computing node**. We use a standard many-core CPU-GPU machine. For this machine we have: one processor Intel® Core™ i7-3720QM

@2.60GHz. (4 physical cores). And one accelerator NVIDIA® GeForce® GTX 670M with 1.3 GB of RAM (336 CUDA cores).

2) **High Performance Computing node**. We use a machine that supports 16 nodes, each one with 8 GPUs [14]. We use one node with following features: two processors Intel® Xeon® E5645 @2.40GHz. (2x6 physical cores). And eight accelerators NVIDIA® Tesla™ M2075 with 4.9 GB of RAM (8x448 CUDA cores).

We have created several scenarios for each machine, based on the memory consumption on the GPU (in-core and out-of-core) and the number of parallel tasks. The first example is based on a Cartesian mesh of an average of 2 million grid points ($160\times160\times80$) in order to fits in the memory available on the GPU of both machines. For the out-of-core example, requiring several data transfers between global and GPU RAM, we have selected two different sizes of problem.
For the commodity computing node we use a three-dimensional grid of size ($640\times640\times80$) points and a grid of size ($1000\times1000\times80$) for the High Performance Computing node. The other parameters (number of tasks for instance) strongly depends on the blocksize variable described in section IV. Table I presents memory consumption, number of blocks and parallel tasks for blocksize=256. The STARPU_CUDA_ASYNC flag enables concurrent kernel

	In-core	Out-of-core	
		Commodity node	HPC node
Memory (MB)	179	2711	6577
Number of blocks	1	9	16
Computation Tasks	40	200	600
Communication Tasks	0	424	1800

Table I: Memory consumption, number of blocks and number of parallel task for the simulated scenarios

execution on graphics cards that support this feature [13]. In our case, this flag is available on both machines. We perform our experiments using several configurations in terms of processing cores usage. For pure GPU experiments, the model is simulated only using the GPU cores available on the target architecture. Symmetrically, the pure CPU executions only target x86 cores. The hybrid experiments tackle all the computing cores available (CPU and CPU cores). Using one GPU, three and eleven CPU cores are respectively used on the commodity platform and on the HPC node (one CPU core is dedicated for the management of each GPU). MultiGPU simulations exploit eight GPU cards and four CPU cores on the larger platform.

VI. Results

To illustrate the complexity of task-based scheduling on heterogeneous architectures, we represent on Figure 4 the distribution of the computing load using three CPU cores and one GPU. The red and the green squares represent the velocity and the stress computing kernels, possibly at different timesteps, and the blue line illustrates data dependency between the subdomains. We can observe the speedup ratio disparity between CPU or GPU computing tasks with size of the squares. Moreover, significant amount of the total elapsed time could correspond to idle time depending on the granularity of the problem and the scheduling strategy.

A. In-core results

In this section, we analyze the overall performance of our methodology considering a problem size that fit in the GPU memory. Obviously this situation is not the most suitable to benefit from the heterogeneous implementation as the price to pay for data transfers between the CPU and the GPU may overcome expected gains from the additional CPU cores.
Table II shows the results obtained on the commodity

	pure GPU	hybrid (dmdar)	hybrid (ws)
Speedup	$\times5.03$	$\times5.02$	$\times0.42$

Table II: Speedup on the commodity-based hardware configuration (in-core dataset)

computing node (the four CPU cores elapsed time is the baseline). Firstly, we can notice that the speedup measured with one GPU over four CPU cores ($\times5.03$) is in the same order of magnitude of the results reported in [9]–[11]. In this case the overhead coming from StarPU runtime system is limited as the number of blocks is very small and the computing kernels are solely scheduled on the GPU.
Hybrid simulations are supposed to exploit both GPU and CPU cores with the DMDAR scheduler that takes into account the cost of data transfer. Indeed, this strategy schedules all the computing tasks on the GPU because of the high cost of data transfers. As a results, we observe similar level of performance compare to the pure GPU implementation. In order to force the usage of both type of cores, we change the scheduling policy using the work-stealing algorithm. This strategy simply implement a situation where idle workers steal work from busy workers. The results is very poor as the original elapsed time is almost multiplied by a factor 2.3.

Figure 5 shows the results for the large computing node (the twelve cores results are the baseline). Similarly to the previous results, the speedup measured with one GPU appears consistent with the scientific literature ($\times6.94$). The impact of the Tesla card available on the HPC node is significant as we observe a ratio of 4.27 between the pure GPU results on these two architectures. Previous remarks on the DMDAR and the work-stealing algorithm remain also valid. In this case, we notice a stronger degradation of the speedup when using all the CPU cores compare to previous experiments. This probably could be explained by the higher level of performance of the GPU on this machine. The situation is even worst when we use eight GPU and four CPU cores. The elapsed time is increasing by more than a factor eleven due to the data transfers and the poor usage of the available resources. The granularity of the problem also plays an important role in this degradation.

B. Out-of-core results

This section discusses the results on heterogeneous architectures when the size of the data exceeds the memory available on the GPU. In this case, the accelerators are fully used as an additional computing resources to the computing power delivered by the CPU cores. Data transfers is of

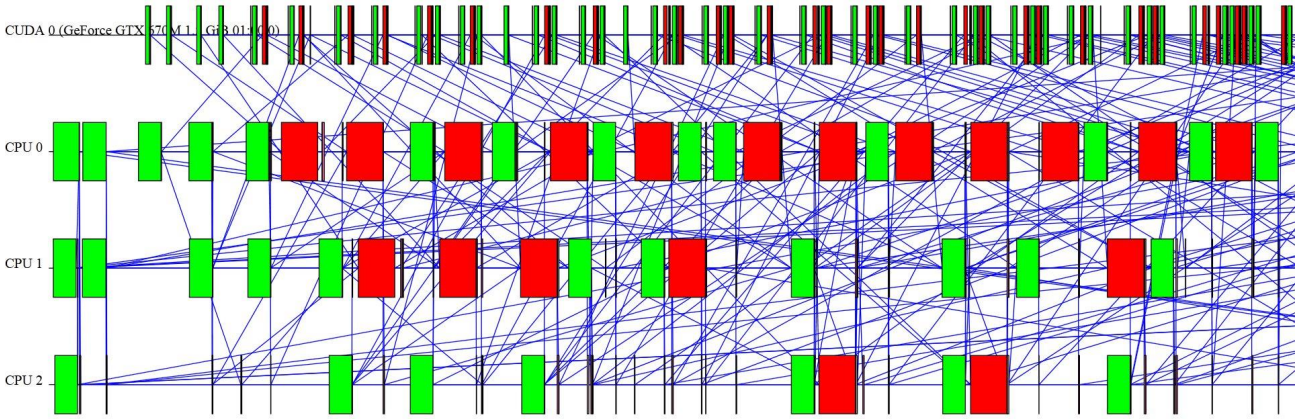


Figure 4: Gantt chart on the commodity-based architecture using three CPU cores and one GPU

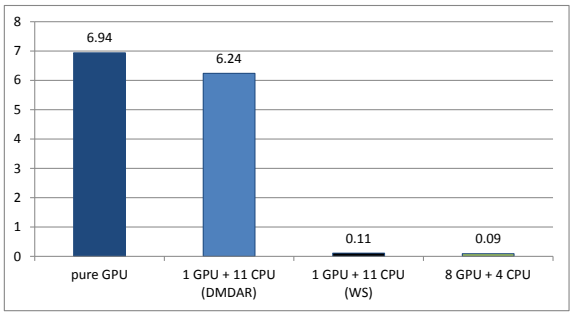


Figure 5: Speedup on the HPC node (in-core dataset)

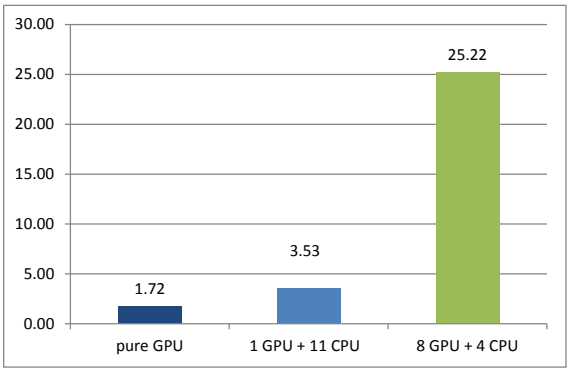


Figure 6: Speedup for out-of-core dataset when running on the HPC node.

great importance and should be carefully controlled by the scheduling strategies.

	pure GPU	hybrid (4 CPU cores and 1 GPU)
Speedup	$\times 0.92$	$\times 1.32$

Table III: Speedup on the commodity-based configuration (out-of-core dataset)

We detail the performance obtained on the commodity node in Table III. We use the four CPU cores results as a baseline. With one GPU, the simulation is slowed down in comparison with the baseline configuration. In this case, we pay the price of data movements between the CPU and the GPU main memory. Indeed, the idle time ratio for the GPU cores reach a maximum of 80.16% preventing any acceleration over the pure CPU version.
The heterogeneous results are slightly better with a speedup of $\times 1.32$. The best results are obtained with the DMDAR scheduler that takes into account the cost of data transfers. Contrary to the results obtained with the in-core dataset, the fine-grained of the decomposition of the large problem (at least 624 tasks) allows to schedule the computing load on all the the computing resources available. Nevertheless, the usage of the cores remains low. The idle time for the three CPU cores varies between 41% and 79%, for the GPU we measure a maximum value of 79%.

We summarize the results on the HPC node in Figure 6. All configurations show a speedup over the baseline results. For the pure GPU experiments, the acceleration reported ($\times 1.72$) demonstrates the benefits from task-based programming even when only the GPU card is used. In comparison with the commodity-based architecture, we benefit from larger bandwidth at the I/O bus level. This value is still far from the average acceleration reported when the problem fits in the GPU memory ($\times 6.94$).
Combining one GPU and the eleven remaining CPU cores leads to an increase of the speedup compared with the GPU only configuration. In this case, our implementation is able to smoothly benefits from this additional computing power by almost doubling the performance level obtained with one GPU. Using both GPU and CPU cores could provide more flexibility for the scheduling of the computing tasks. As a side effect, we also observe a better usage of the GPU resources. The multi-GPU results confirm this trend with a maximum speedup of 25.22. Considering the various parameters that need to be taken into account, any tentative of exhaustive scalability analysis would be false. Indeed, this result should be compare to the corresponding speedup for in-core dataset with one GPU ($\times 6.94$) to understand the remaining effort to fully optimized our implementation on multi-GPU.

C. Impact of the tuning parameters

Several parameters may significantly influence the performance of our task-based implementations of the seismic wave numerical kernel. In this section we underline the impact of the granularity but also of the scheduling algorithms. This section puts the stress on two major features of our algorithm but several other parameters (calibration of StarPU, NUMA effects, shape of the subdomains, scheduling overhead) may also be critical.

1) Size of the block: In this section, we show that selecting the best granularity is crucial to maximize the performance. Considering heterogeneous platforms, this parameter is of great importance. Figure 7 shows the speedups over one CPU core for in-core and out-of-core data set. The results have been obtained on the commodity node. We can observe that depending on the hardware configuration, the most efficient granularity may vary. In both cases, the efficiency with one GPU is optimal with a size of block of 256. When we decrease the size of the block the GPU efficiency is significantly reduces (from $\times 17.3$ to $\times 4.3$ for in-core problems). This is coming from the GPU architecture that could deliver an optimal level of performance when all vector units are used. This means that larger block perform better on such architecture. The situation is rather different on CPU platforms as tiny blocks could improve locality and cache effects. In this case, using a large number of blocks is also mandatory to extract enough concurrency from our task-based algorithm. Indeed, our wavefront decomposition reach a good level of efficiency with blocks of size equal to 64 or less for the in-core problem. For the out-of-core problem, we generate enough tasks to benefit from the four CPU cores in all cases. The bigger problem size corresponding to more computing tasks explains this result.
Indeed, the choice of the appropriate granularity relies on a tradeoff between the performance level of the computing tasks depending on their sizes and the suitable number of tasks necessary to provide enough parallelism. Obviously, the optimal configuration is not always possible depending on the overall size of the domain and speedup ratio between the processing units.

2) Scheduling strategies: One of the key contribution of StarPU runtime system is to provide several scheduling algorithms adapted to various computing load and hardware heterogeneity. In this section, we compare several different schedulers. Figure 8 shows the speedup over the worst result on each platform. On commodity-based platform, we consider one GPU and three CPU cores. In this case the best results are provided by the DMDAR algorithm that takes into account the cost of data transfers. Eager and Work Stealing algorithm appear like alternatives to DMDAR. These two algorithms do not use information from the memory bus. The rather good results underline the limited contention at the memory bus level for this configuration.
The situation is rather different on the HPC node. The best results are also obtained with the DMDAR algorithm but the performance with the other schedulers is very poor. In this case we use eight GPU and four CPU cores and data transfers optimization is critical, especially for multi-accelerator platforms with major bottlenecks.

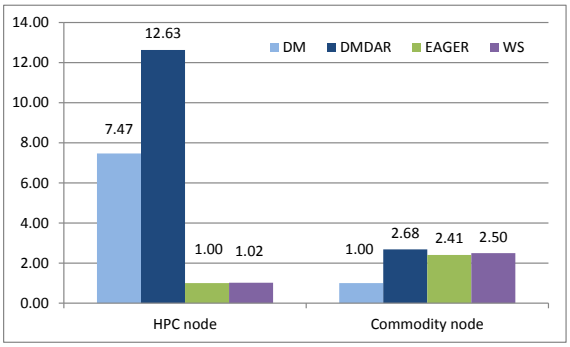
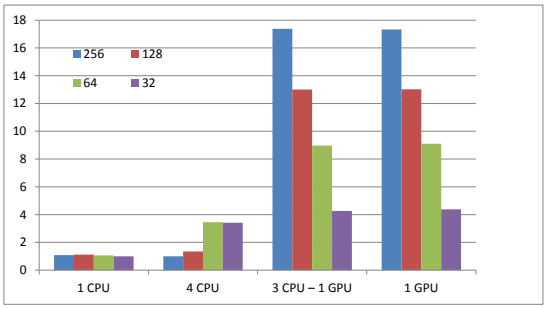


Figure 8: Impact of the scheduling algorithms for experiments on heterogeneous platforms. Relative speedup over the worst situation on each platform.

VII. Related works

Several references implement scientific applications on heterogeneous architectures, but most of these works don't exploit all the computing resources available or depend on a low-level programming approach. Linear algebra is the standard domain that exploits task-based programming model. This is coming from from the Blas-based formulation available for most standard solvers (*i.e.* LU, QR). In [15], [16], Agullo *et al.* implement LU and QR decomposition algorithm for heterogeneous architectures exploiting task-based parallelism on top of StarPU runtime system. A comparison between StarPU and Parsec runtime systems is provided in [17], the authors conclude that both runtime systems are able to benefit from heterogeneous platforms with comparable levels of performance.
If we focus on seismic wave propagation numerical kernels, a large body of the recent scientific literature is devoted to the adaptation of this algorithm on homogeneous multicore architectures. This is coming from the versatility of the Finite Difference Method (FDM) implemented both in the oil and gas industry and also for earthquakes modeling. Several strategies have been proposed mainly to optimize CPU or GPU implementations.
On x86 architecture a review of scalability issues could be found in [8]. Recent efforts are mainly devoted to low-level optimizations including auto-tuning strategies to fully benefits from vectorization [18], [19]. On Graphics Processing Unit (GPU) reference implementations that optimize flops/bytes ratio and reduce data transfers are described in [9], [12]. Finally, Intel and NVIDIA have introduced highly-tuned versions of such kernels [11], [20].
One can also underline relevant references that tackle energy-efficiency issues considering a co-desing strategy [21], many-core embedded architecture [22] or efficient numerical space-time decomposition based on the parareal method [23].
Recent contributions have been made to improve seismic wave modeling on heterogeneous platforms. In [24], the authors implement a seismic model using task-based programming but they don't use heterogeneous cores for the simulations. Calandra *et al.* [25] evaluate execution of a finite-differences stencil on CPUs, APUs and GPUs but they don't combine heterogeneous cores, neither parallel tasks programming.



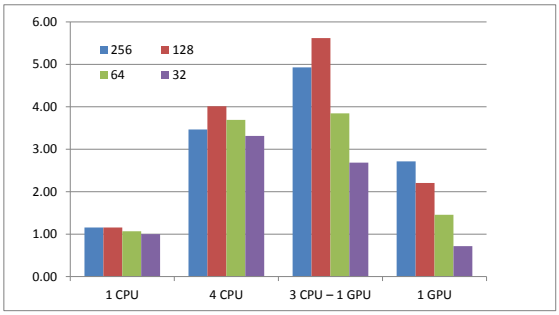


Figure 7: Impact of the granularity on the efficiency of seismic wave modelling on GPUs+CPUs. In-core results (left) and the out-of-core results (right).

VIII. Conclusion and Perspectives

In this paper, we have presented a task-based implementation of the elastodynamics equation. Revisiting the standard data-parallelism arising from finite-differences numerical method, we define a graph of tasks corresponding to the fourth-order stencil formulation. We therefore benefit from StarPU runtime system in order to smoothly schedule this DAG (Directed Acyclic Graph) on current heterogeneous platform. In this way, we decouple the numerical algorithm and the underlying architecture by fully exploiting the versatility of modern runtime systems. This approach allows us to tackle the complexity of this memory-bound problem by minimizing data movements to and from the accelerators.

We demonstrate the efficiency of our approach on a two heterogeneous architectures using the maximum number of processing units available. Considering realistic problem that could not fit in the GRAM memory, we observe a maximum speedup of 25.22 using four CPU cores and eight GPU in comparison with the same experiments using twelve CPU cores. Using a commodity-based architecture with four CPU cores and one GPU, we measure an acceleration of 32% over the CPU version. The analysis of the performance underlines the significant impact of the granularity and the scheduling strategy.

These promising first steps open several promising ways to optimize the efficiency of our approach on heterogeneous platform. Firstly, a detailed comprehension of the tradeoff between the granularity and the scheduling policies is necessary in order to tackle more complex architecture. For instance, the regularity of the finite-differences numerical method could allow us to derive a cost-model that could help the runtime system to improve task scheduling. This includes potential irregularity in the granularity of the task to maximize efficiency on multiple devices. Secondly, we plan to extend the implementation of our elastodynamics equation by including absorbing boundary conditions [26]. This additional layer generates load imbalance that could be smoothly reduced by the fine-grained task-based programming model. Finally, the emerging of integrated cores architectures that deliver higher bandwidth between GPU and CPU appears like an opportunity to tackle both the PCI-express bottleneck and the energy-efficiency challenge.

Acknowledgment

This work have been supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES). Experiments presented in this paper were carried out using the GUANE-1 experimental testbed, being developed under the Universidad Industrial de Santander (SC3UIS) High Performance and Scientific Computing Centre, development action with support from UIS Vicerrectoría de Investigación y Extensión (VIE-UIS) and several UIS research groups as well as other funding bodies (see http://grid.uis.edu.co and http://www.sc3.uis.edu.co).

References

[1] R. Vasudevan, S. S. Vadhiyar, and L. V. Kalé, "G-charm: an adaptive runtime system for message-driven parallel applications on hybrid systems," in *International Conference on Supercomputing, ICS'13, Eugene, OR, USA - June 10 - 14, 2013*, 2013, pp. 349–358.

[2] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Luszczek, S. Tomov, and J. Dongarra, "Scalable dense linear algebra on heterogeneous hardware," *Advances in Parallel Computing*, 2013.

[3] J. Bueno, X. Martorell, R. M. Badia, E. Ayguadé, and J. Labarta, "Implementing ompss support for regions of data in architectures with multiple address spaces," in *International conference on Supercomputing*, 2013.

[4] T. Gautier, J. V. F. Lima, N. Maillard, and B. Raffin, "Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures," in *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, 2013, pp. 1299–1308.

[5] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011.

[6] J. Virieux, "P-SV wave propagation in heterogeneous media; velocity-stress finite-difference method," *Geophysics*, vol. 51, no. 4, pp. 889–901, 1986.

[7] P. Moczo, J. Robertsson, and L. Eisner, "The finite-difference time-domain method for modeling of seismic wave propagation," in *Advances in Wave Propagation in Heterogeneous Media*, ser. Advances in Geophysics. Elsevier - Academic Press, 2007, vol. 48, ch. 8, pp. 421–516.

[8] F. Dupros, H. Do, and H. Aochi, "On scalability issues of the elastodynamics equations on multicore platforms," in *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, pp. 1226–1234.

[9] R. Abdelkhalek, H. Calandra, O. Coulaud, G. Latu, and J. Roman, "Fast seismic modeling and reverse time migration on a graphics processing unit cluster," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 7, pp. 739–750, 2012.

[10] L. de Oliveira Martins, M. A. G. da Silva, M. Arruda, J. Duarte, P. M. Silva, R. B. Seixas, and M. Gattass, "Accelerating curvature estimate in 3d seismic data using GPGPU," in *26th IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2014, Paris, France, October 22-24, 2014*, 2014, pp. 105–111.

[11] P. Micikevicius, "3D finite-difference computation on GPUs using CUDA," in *Workshop on General Purpose Processing on Graphics Processing Units*. Washington, USA: ACM, 2009, pp. 79–84.

[12] D. Michéa and D. Komatitsch, "Accelerating a 3D finite-difference wave propagation code using GPU graphics cards," *Geophysical Journal International*, vol. 182, no. 1, pp. 389–402, 2010.

[13] U. de Bordeaux, CNRS, and INRIA, "Starpu handbook," http://starpu.gforge.inria.fr/doc/starpu.pdf, 2014.

[14] U. I. de Santander, "Super computación y cálculo científico UIS," http://www.sc3.uis.edu.co/servicios/hardware/, 2015.

[15] E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, J. Langou, H. Ltaief, and S. Tomov, "LU factorization for accelerator-based systems," in *Proceedings of the 2011 9th IEEE/ACS International Conference on Computer Systems and Applications*, ser. AICCSA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 217–224.

[16] E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, S. Thibault, and S. Tomov, "QR factorization on a multicore node enhanced with multiple GPU accelerators," in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011 - Conference Proceedings*, 2011, pp. 932–943.

[17] X. Lacoste, M. Faverge, G. Bosilca, P. Ramet, and S. Thibault, "Taking advantage of hybrid systems for sparse direct solvers via task-based runtimes," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*, 2014, pp. 29–38.

[18] M. Christen, O. Schenk, and Y. Cui, "Patus for convenient high-performance stencils: evaluation in earthquake simulations," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. IEEE Computer Society Press, 2012, pp. 11:1–11:10.

[19] H. Dursun, K.-I. Nomura, L. Peng, R. Seymour, W. Wang, R. K. Kalia, A. Nakano, and P. Vashishta, "A Multilevel Parallelization Framework for High-Order Stencil Computations," in *Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference*, Delft, The Netherlands, August 2009, pp. 642–653.

[20] J. Reinders and J. Jeffers, *High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches*, 1st ed. Morgan Kaufmann, 2014.

[21] J. Krueger, D. Donofrio, J. Shalf, M. Mohiyuddin, S. Williams, L. Oliker, and F.-J. Pfreund, "Hardware/software co-design for energy-efficient seismic modeling," in *Supercomputing Conferece (SC)*. IEEE Computer Society, 2011, pp. 73:1–73:12.

[22] M. Castro, F. Dupros, E. Francesquini, J.-F. Mehautk, and P. Navaux, "Energy efficient seismic wave propagation simulation on a low-power manycore processor," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on*, Oct 2014, pp. 57–64.

[23] D. Mercerat, L. Guillot, and J.-P. Vilotte, "Application of the Parareal Algorithm for Acoustic Wave Propagation," in *American Institute of Physics Conference Series*, ser. American Institute of Physics Conference Series, T. E. Simos, G. Psihoyios, and C. Tsitouras, Eds., vol. 1168, Sep. 2009, pp. 1521–1524.

[24] L. Boillot, G. Bosilca, E. Agullo, and H. Calandra, "Task-based programming for seismic imaging: Preliminary results," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, Aug 2014, pp. 1259–1266.

[25] H. Calandra, R. Dolbeau, P. Fortin, J.-L. Lamotte, and I. Said, "Evaluation of successive CPUs/APUs/GPUs based on an OpenCL finite difference stencil," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, Feb 2013, pp. 405–409.

[26] R. K. Tesser, L. L. Pilla, F. Dupros, P. O. A. Navaux, J. Méhaut, and C. L. Mendes, "Improving the performance of seismic wave simulations with dynamic load balancing," in *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, Torino, Italy, February 12-14, 2014*, 2014, pp. 196–203.