Combining recent HPC techniques for 3D geophysics acceleration

Lionel Boillot, Emmanuel Agullo, George Bosilca, Henri Calandra

YIC2013
Second ECCOMAS Young Investigators Conference
2–6 September 2013, Bordeaux, France

Combining recent HPC techniques for 3D geophysics acceleration

L. Boillot [a,*], E. Agullo [b], G. Bosilca [c], H. Calandra [d]

[a] INRIA EPI Magique-3D - LMA, Pau, France
INRIA Bordeaux Sud-Ouest, Magique-3D project team, Université de Pau et des Pays de l'Adour, Avenue de l'Université, BP 1155, 64013 Pau

[b] INRIA EPI HiePACS, Talence, France
INRIA Bordeaux Sud-Ouest, HiePACS project team, 200 avenue de la Vieille Tour, 33405 Talence cedex, France

[c] Innovative Computing Laboratory, University of Tennessee, Knoxville, USA
ICL, University of Tennessee, Department of Electrical Engineering and Computer Science, 1122 Volunteer Blvd, Knoxville TN 37996

[d] TOTAL Exploration & Production PAU, France
TOTAL EP, Depth Imaging and High Performance Computing, Avenue Larribau, 64018 Pau

[*] lionel.boillot@inria.fr

Abstract. *Reverse Time Migration technique produces underground images using wave propagation. A discretization based on the Discontinuous Galerkin Method unleashes a massively parallel elastodynamics simulation, an interesting feature for current and future architectures. In this work, we propose to combine two recent HPC techniques to achieve a high level of efficiency: the use of runtimes (StarPU and PaRSEC) to easily exploit the hardware capabilitites, and the intergration of accelerators (Intel Xeon Phi). Preliminary results are presented.*

Keywords: elastodynamics; Discontinuous Galerkin Method; accelerator; runtime; High-Performance Computing

1 INTRODUCTION

Reverse Time Migration (RTM) is one of the most widely used technique of Seismic Imaging. It is a particular instance of the depth migration algorithm, a technique that can be successfully applied in areas with lateral velocity variations, as those of main interesting to petroleum geologists. Thanks to the space discretization by the Discontinuous Galerkin Method (DGM), associated with a leap-frog time scheme, the numerical problem becomes essentially local. Besides, it generates a massively amount of computations with inherent parallelism, an interesting property to take advantage of, on the current and future, heterogeneous many-cores architectures.

While, the use of supercomputers is mainstream in all scientific areas, efficiently exploiting the available hardware remains a challenging task. The increasing level of hierarchization of the memory and the hybridization of the underlying hardware, force the programmers to develop and refine specialized algorithms, so as to expose an adequate degree of parallelism which encompasses all available resources, including accelerators. This highlights the limits of current programming paradigms to deal with this raise in heterogeneity, and as a result the portability of scientific applications is strongly impacted, lower the productivity in the long term.

In this context, the use of programming paradigms based on task-graph approaches allows deep parallelism automatic scheduling over dynamic runtimes. This sets up the hope for a stable and portable efficiency of algorithms, expressed in this paradigm, across generation of architectures. In this paper we will explore this approach applied to elastodynamics simulation.

2 ELASTODYNAMICS

Let Ω be an open bounded domain of $\mathbb{R}^3$, its boundary is denoted by $\Gamma = \partial\Omega$ with the exterior normal $\mathbf{n_\Gamma}$. Let $\mathbf{x} = (x, y, z) \in \Omega$ and $t \in [0, T]$ be the space and time variables, the velocity-stress formulation of the elastic wave equation is

$$\begin{cases} \rho(\mathbf{x})\partial_t v(\mathbf{x},t) &= \nabla.\underline{\underline{\sigma}}(\mathbf{x},t) \\ \partial_t \underline{\underline{\sigma}}(\mathbf{x},t) &= \underline{\underline{C}}(\mathbf{x}) : \underline{\underline{\epsilon}}(v(\mathbf{x},t)) \end{cases} \tag{1}$$

with $\rho > 0$ the volumic mass, $v \in \mathbf{H}^1(\Omega \times [0,T])$ the unknown velocity field, $\underline{\underline{\sigma}} \in \underline{\underline{H}}_{div}(\Omega \times [0,T])$ the stress tensor, $\underline{\underline{C}}$ the stiffness tensor (elasticity coefficients) and $\underline{\underline{\epsilon}}(v) = \frac{1}{2}(\boldsymbol{\nabla} v + (\boldsymbol{\nabla} v)^T)$ the strain tensor (infinitesimal strain theory) with ∇ the divergence and $\boldsymbol{\nabla}$ the gradient.

2.1 Discretization

Let Ω_h be a polygonal mesh (composed of tetrahedra K in 3D) forming a partition of Ω. Contrary to FEM, the principle of DGM consists in approximating the functions v and $\underline{\underline{\sigma}}$ by discontinuous functions v_h and $\underline{\underline{\sigma}}_h$ such that $\{v_h, \underline{\underline{\sigma}}_h\} \in L^2(\Omega_h \times [0,T])$ with $\{v_h|_K, \underline{\underline{\sigma}}_h|_K\} \in H^1(K \times [0,T])\ \forall K$. Thus, the integration against discontinuous test functions w and $\underline{\underline{\xi}}$ can be performed locally on each tetrahedron. Then, the sum can be written as

$$\begin{cases} \sum\limits_K \int_K \rho_K \partial_t v w d\mathbf{x} = \sum\limits_K \left(\int_{\Gamma_K} \{\{\underline{\underline{\sigma}}\mathbf{n_K}\}\}.[\![w]\!] d\mathbf{x} - \int_K \underline{\underline{\sigma}} : \boldsymbol{\nabla} w d\mathbf{x} \right), \\ \sum\limits_K \int_K \partial_t \underline{\underline{\sigma}} : \underline{\underline{\xi}} d\mathbf{x} = \sum\limits_K \left(\int_{\Gamma_K} [\![(\underline{\underline{C}} : \underline{\underline{\xi}})\mathbf{n_K}]\!]\{\{v\}\} d\mathbf{x} - \int_K v.\nabla.(\underline{\underline{C}} : \underline{\underline{\xi}}) d\mathbf{x} \right), \end{cases} \tag{2}$$

with the centered fluxes formulation: $\{\{u\}\} = (u_{K_1} + u_{K_2})/2,\ [\![u]\!] = u_{K_1} - u_{K_2}$.

These equations can be written as a linear system,

$$\begin{cases} \rho M_v \partial_t v_h + R_{\underline{\underline{\sigma}}} \underline{\underline{\sigma}}_h &= 0 \\ M_{\underline{\underline{\sigma}}} \partial_t \underline{\underline{\sigma}}_h + R_v v_h &= 0 \end{cases} \tag{3}$$

with $M_v,\ M_{\underline{\underline{\sigma}}}$ block-diagonal mass matrices and $R_v,\ R_{\underline{\underline{\sigma}}}$ stiffness matrices.

The time discretization is done with the classical leap-frog time scheme, leading to an explicit scheme which is stable under a CFL condition available in [1]. Time domain $[0,T]$ is divided into time steps Δt. Let v_h^n be the approximation of the velocity $v_h(t)$ at the discrete time $t = n\Delta t$ and $\underline{\underline{\sigma}}_h^{n+1/2}$ the approximation of the stress tensor $\underline{\underline{\sigma}}_h(t)$ at the discrete time $t = (n + \frac{1}{2})\Delta t$. Semi-discrete linear system (3) becomes

$$\begin{cases} \rho M_v \dfrac{v_h^{n+1} - v_h^n}{\Delta t} + R_{\underline{\underline{\sigma}}} \underline{\underline{\sigma}}_h^{n+1/2} &= 0 \\ \\ M_{\underline{\underline{\sigma}}} \dfrac{\underline{\underline{\sigma}}_h^{n+3/2} - \underline{\underline{\sigma}}_h^{n+1/2}}{\Delta t} + R_v v_h^{n+1} &= 0 \end{cases} \tag{4}$$

Since M_v and $M_{\underline{\underline{\sigma}}}$ are block-diagonal due to DGM, this scheme is quasi-explicit. Indeed, each block has the size of the cell (i.e. the number of degrees of freedom, see Figure 1 for a 2D illustration), allowing its direct inversion.

3 HPC OPTIMIZATIONS

Several approaches have been investigated in the context of this work, starting from a reordering of the data to maximize the cache locality, re-describing the underlying algorithms using novel programming techniques and, finally, including the usage of accelerators.

Cache optimization/reordering and vectorization The overall efficiency is strongly correlated to the efficiency of the most used part of the algorithm. A poorly-parallelized detail in the code would lead to a severe bottleneck.

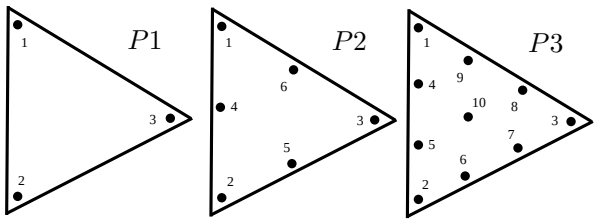

Figure 1: DGM local degrees of freedom in 2D

A dissection of the original code revealed several cache under-utilization problems and a lack of vectorization in the main computational algorithms. The accesses to the data were scattered in the memory, leading to significantly more cache misses than necessary. Moreover, the communications moved too much data to and from the memory according to what was really needed. A major reordering of the loops and memory accesses have allowed the compilers to expose more vectorization possibilities and to realize a better instruction level scheduling. This involves a direct result of a significantly improved processing time, as highlighted in Table 1.

The vectorization point is particularly essential for the use of Intel Xeon Phi accelerators, which are based on large SIMD (Single Instruction Multiple Data) units. The cache-reuse is a more general common source of optimizations which always offers good acceleration.

Task-based description for the use of runtimes Runtimes, like StarPU [2] (Inria, Bordeaux, France) and PaRSEC [3] (ICL, University of Tennessee, USA), offer automatic parallelism which maximize the performance of the algorithm while hiding the cluster complexity (heterogeneity, NUMA, network, etc). The principle lies in describing the algorithm using a task-based description (computations, communication, dependencies), and then, allowing the runtime to choose where and when to schedule these tasks in order to minimize the waiting time, to maximize the overlap between communications and computations, and to maximize the utilization of all heterogeneous resources.

Heterogeneity (accelerators) with the same code kernels GPGPU suffers from several critical points (new language, long formation-time, etc) but can improve significantly numerous codes. The current programming approaches are too tailored for this specific hardware (CUDA, OpenCL), with a lack of long term vision regarding the performance portability and the implementation perennially. The new Intel accelerator, Xeon Phi [4], is based on a mix of CPU and GPU architectures and seems to have multiple advantages. First it is based on the x86 instruction set architecture, this means that it can manage the classical languages C, C++, FORTRAN. Next, it can be viewed as a little standalone cluster, allowing the use of classical parallel tools and libraries as OpenMP and MPI. The idea here is to use accelerator capabilities through only one optimized version of the code kernels.

3.1 Preliminary results

In order to use both the Intel accelerator and the runtimes capabilities, we began by adding OpenMP in the original code. This did not bring acceleration compared to full MPI parallelism but it will be very useful for the future benchmarks, especially with the combined HPC techniques. Then we redesigned the main algorithms correcting the above listed issues. The solution involved a little bit more computations and less communications. Nevertheless, the simulation time decreased seriously with two-fold acceleration on average. The Table 1 presents a range of results on 8 processors with different MPI-OpenMP distribution over various discretization configurations.

Table 1: Speedup between the original code and the optimized code

Distribution	**2D–P1**	**2D–P2**	**2D–P3**	**3D–P1**	**3D–P2**	**3D–P3**
1 mpi, 8 omp	2.46	2.44	2.17	3.24	2.51	2.40
2 mpi, 4 omp	2.47	2.24	2.15	2.74	2.13	2.09
4 mpi, 2 omp	2.41	2.19	2.15	2.49	1.93	1.81
8 mpi, 1 omp	2.57	2.16	2.13	2.41	1.70	1.76

3.2 Scalability benchmark

A scalability study was performed on a 3D case. The unstructured meshes represent one homogeneous subsurface layer. The mesh size (in number of tetrahedra) begins at 50.000 and stops to 800.000. As the number of time-steps depends on the element size, the code is voluntarily stopped after 5.000 iterations, in order to allow the comparison. The number of CPUs starts from 4 and goes up to 256 so as to ensure that in the range of mesh sizes, the volume of computations is enough on the one hand, and the communications are dominant on the other hand. Moreover, a run with only one CPU is done to catch reference time for the efficiency computation. Figure 2 confirms that the redesigned code achieves a good parallel efficiency ($> 90\%$ and even $> 95\%$) as soon as the mesh size is large enough according to the number of CPUs.

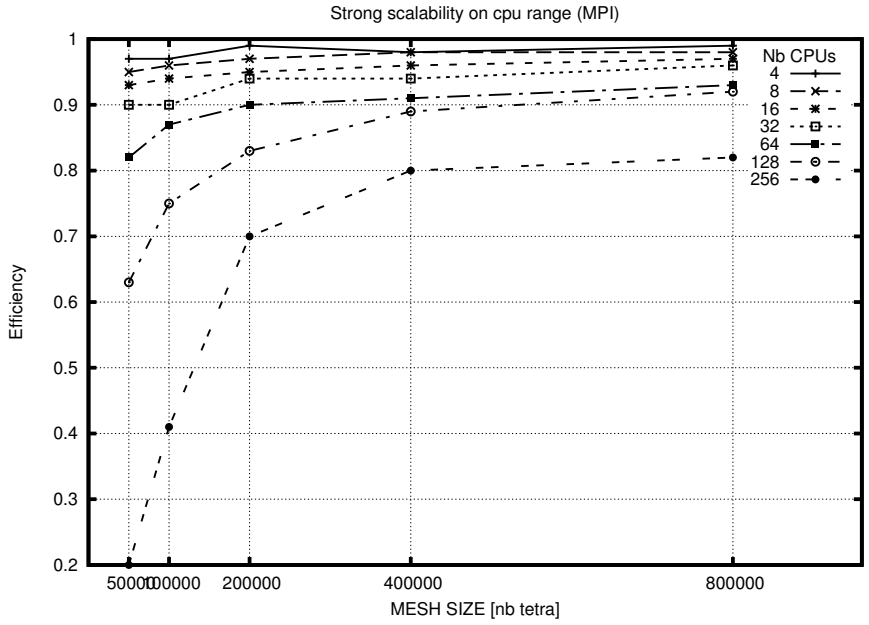

Figure 2: Efficiency results of the scalability study

4 CONCLUSIONS

The first step of optimizing the elastodynamics simulation code provided a two-fold acceleration and a good scalability. These modifications laid down a cleaner path to move the code over to a task-based programming paradigm and to benefit from dynamic runtimes. This work is still ongoing. Simultaneously, specialized version of the computational code kernels will be implemented to take advantage of the Intel Xeon Phi accelerators, supported by these runtimes. The final purpose is to combine the two HPC techniques allowing automatic parallelism and efficiency on heterogeneous clusters from a unique optimized code.

ACKNOWLEDGEMENT

The authors acknowledge the support by the INRIA-TOTAL strategic action DIP (`http://dip.inria.fr`).

REFERENCES

[1] Delcourte, S.; Fezoui, L. & Glinsky-Olivier, N. A high-order discontinuous Galerkin method for the seismic wave propagation. *ESAIM: Proceedings*, **27**:70-89, 2009

[2] Augonnet, C.; Thibault, S.; Namyst, R. & Wacrenier, P.-A. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency Computat.: Pract. Exper.*, **23**: 187-198, 2011

[3] Bosilca, G.; Bouteiller, A.; Danalis, A.; Herault, T.; Lemarinier, P. & Dongarra, J. DAGuE: A generic ditributed DAG engine for High Performance Computing. *Parallel Computing*, **38**:37-51, 2012

[4] Reinders, J. An overview of programming for Intel Xeon processors and Intel Xeon Phi coprocessors. *Intel*, 2012