MADRID 2015
1-4 June 2015 I IFEMA Madrid

Th N112 13

Accelerated Discontinuous Galerkin Time-domain Simulations for Seismic Wave Propagation

A. Modave* (Rice University), A. St-Cyr (Shell GSI BV), T. Warburton (Rice University) & W.A. Mulder (Shell GSI BV & Delft University of Technology)

SUMMARY

Improving both the accuracy and computational performance of simulation tools is a major challenge for seismic imaging, and generally requires specialized algorithms and computational implementations to make full use of modern hardware architectures. We present a computational strategy based on a high-order discontinuous Galerkin time-domain method. Our implementation can be run on several architectures thanks to a unified multi-threading programming framework, and exhibits a good load balancing and minimum data movements. A first benchmark validates this implementation and confirms the interest of accelerators in computational geophysics.

Introduction

Improving both the accuracy and computational performance of numerical tools is a major challenge for geophysical applications, which are very demanding in computational resources. The current generation of clusters consist of many-core CPU and optionally massively parallel graphics processing units or side-car accelerators to provide a performance boost. However, accelerator-aided clusters require specialized algorithms and computational implementations to make full use of the hardware.

High-order discontinuous Galerkin (DG) finite-element time-domain methods exhibit good features for computations on such modern hardware architectures. The weak element-to-element coupling and the dense algebraic operations required per element make them suitable schemes for parallel multi-threading computations, especially with graphics processing units (GPU) (Klöckner et al., 2009; Mu et al., 2013).

We are developing a computational tool for seismic imaging based on a nodal high-order penalty DG method. Our implementation utilizes OCCA (Medina et al., 2014), a unified programming framework that abstracts major multi-threading languages (OpenCL, CUDA, pThreads, and OpenMP), offering flexibility to choose the hardware architecture and thread-model at run-time. In this paper, some key elements of both the numerical scheme and our computational implementation are explained in a fundamental case. A validation case is presented with computational results obtained using a single GPU with CUDA and OpenCL through the OCCA framework.

Method

In this paper, we deal with the acoustic wave model based on the first-order system

$$\frac{\partial p}{\partial t} + \rho c^2 \nabla \cdot \mathbf{v} = 0, \tag{1}$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \nabla p = 0, \tag{2}$$

in a general three-dimensional domain $\Omega \subset \mathbb{R}^3$. The fields $p(\mathbf{x},t)$ and $\mathbf{v}(\mathbf{x},t)$ represent the pressure and the particle velocity, respectively. The density $\rho(\mathbf{x})$ and the phase velocity $c(\mathbf{x})$ are assumed to be positive and piecewise constant. The boundary condition can be $p = 0$ for a free surface, or an absorbing boundary condition when the domain is artificially truncated.

Numerical Scheme

We seek approximate solutions built on a mesh made of non-overlapping tetrahedra, $\Omega = \bigcup_{k=1,\ldots,K} \mathsf{D}_k$, where K is the number of cells and D_k is the k^{th} cell. The parameters ρ and c are constant over each cell and can be discontinuous at interfaces. With the nodal discontinuous Galerkin method, the pressure field and each Cartesian component of the velocity field are approached by piecewise polynomial functions. The discrete unknowns are associated to the values of fields at nodes of elements (Hesthaven and Warburton, 2007). For each cell D_k, we then have

$$p_k(\mathbf{x},t) = \sum_{n=1}^{N_p} p_{k,n}(t)\, \ell_{k,n}(\mathbf{x}), \qquad \forall \mathbf{x} \in \mathsf{D}_k, \tag{3}$$

$$v_{i,k}(\mathbf{x},t) = \sum_{n=1}^{N_p} v_{i,k,n}(t)\, \ell_{k,n}(\mathbf{x}), \qquad \forall \mathbf{x} \in \mathsf{D}_k, \quad i = 1,2,3, \tag{4}$$

where N_p is the number of nodes by element, $p_{k,n}(t)$ and $v_{i,k,n}(t)$ are the values of fields at node n of element k, and $\ell_{k,n}(\mathbf{x})$ is the associated multivariate Lagrange polynomial function. We consider a

numerical scheme based on the weak form

$$\int_{\mathrm{D}_k}\left(\frac{1}{\rho c^2}\frac{\partial p}{\partial t}+\nabla\cdot\mathbf{v}\right)\varphi(\mathbf{x})\,d\mathbf{x}+\int_{\partial\mathrm{D}_k}\frac{1}{2}\left(\mathbf{n}\cdot[\![\mathbf{v}]\!]-\tau_p[\![p]\!]\right)\varphi(\mathbf{x})\,d\mathbf{x}=0, \tag{5}$$

$$\int_{\mathrm{D}_k}\left(\rho\frac{\partial \mathbf{v}}{\partial t}+\nabla p\right)\cdot\psi(\mathbf{x})\,d\mathbf{x}+\int_{\partial\mathrm{D}_k}\frac{1}{2}\mathbf{n}\left([\![p]\!]-\tau_{\mathbf{v}}\mathbf{n}\cdot[\![\mathbf{v}]\!]\right)\cdot\psi(\mathbf{x})\,d\mathbf{x}=0, \tag{6}$$

for all test functions $\varphi(\mathbf{x})\in P^N(\mathrm{D}^k)$ and $\psi(\mathbf{x})\in\left(P^N(\mathrm{D}^k)\right)^3$ where $P^N(\mathrm{D}^k)$ is the space of polynomials in $\mathbf{x}$ of total degree N. In the boundary terms, $\mathbf{n}$ is the outward unit normal to the cell boundary $\partial\mathrm{D}_k$, and $[\![p]\!]=p^+-p^-$ and $[\![\mathbf{v}]\!]=\mathbf{v}^+-\mathbf{v}^-$ are the jumps of fields, where the plus and minus subscripts denote nodal values on the side of the neighboring and current cells, respectively. We use the stabilization parameters $\tau_p=1/\{\rho c\}$ and $\tau_{\mathbf{v}}=\{\rho c\}$, where the brackets denote the average of coefficients. Substituting the polynomial approximation of fields in the weak form (5)–(6), and using the Lagrange functions as test functions, we obtain a coupled system of ordinary differential equations

$$\frac{d\mathbf{q}_k^p}{dt}=-\rho_k c_k^2\sum_{i=1}^{3}\sum_{j=1}^{3}\frac{\partial\Psi_{k,i}}{\partial r_j}\mathbf{D}_j\mathbf{q}_k^{v_i}-\sum_{f=1}^{4}\frac{J_{k,f}}{J_k}\mathbf{L}_f\mathbf{P}_{k,f}^p, \tag{7}$$

$$\frac{d\mathbf{q}_k^{v_i}}{dt}=-\frac{1}{\rho_k}\sum_{j=1}^{3}\frac{\partial\Psi_{k,i}}{\partial r_j}\mathbf{D}_j\mathbf{q}_k^p-\sum_{f=1}^{4}\frac{J_{k,f}}{J_k}\mathbf{L}_f\mathbf{P}_{k,f}^{v_i},\qquad i=1,2,3, \tag{8}$$

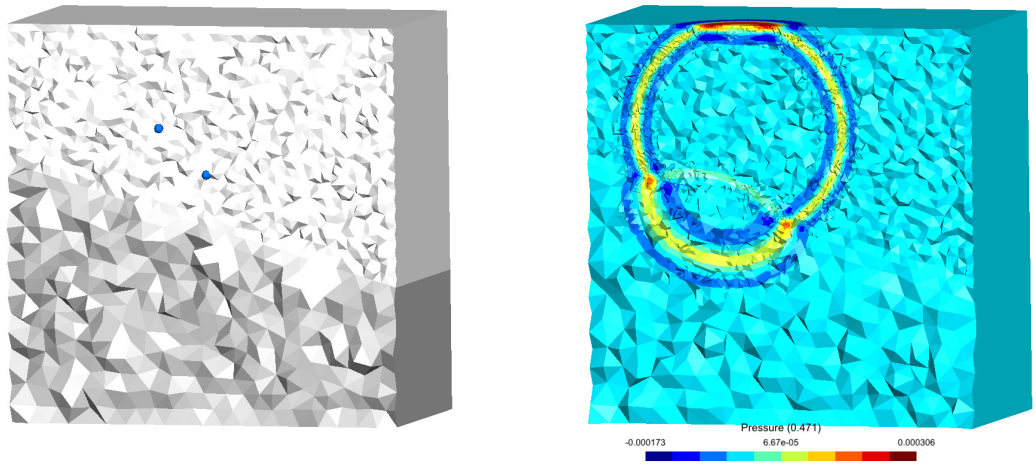
for each element k. The vectors $\mathbf{q}_k^p$ and $\mathbf{q}_k^{v_j}$ contain the discrete unknowns associated to each field for all nodes of element k, and $\mathbf{P}_{k,f}^p$ and $\mathbf{P}_{k,f}^{v_j}$ contain the penalty terms for all nodes of face f of element k. The matrices $\mathbf{D}_j$ and $\mathbf{L}_f$ are respectively the differentiation and lifting matrices for the reference element. The matrices are recovered for the current element thanks to geometric factors based on mapping functions $\Psi_{k,i}(\mathbf{r})$ and Jacobian determinants J_k and $J_{k,f}$ (Hesthaven and Warburton, 2007; Klöckner et al., 2009).

In this work, the time stepping is performed with an explicit third-order Adams-Bashforth scheme. To compute the solution at a given time step, the solution at the previous time step must be known, as well as the value of the right-hand side of equations (7)–(8) at the three previous time steps.

Computational Implementation

Our implementation follows the philosophy of GPU programming. The algebraic operations of the DG scheme are performed directly on the hardware device (e.g. CPU or GPU) by kernel codes. The execution of these kernels is driven by a main process from the host platform. The OCCA library provides a host API with tools to generate a self-contained context and command queue from hardware device, to allocate and manage device memory, and finally to compile and execute kernels. The OCCA device API allows us to write kernels that are compiled at run-time, and that send instructions to the devices when executed. The kernel language of OCCA mirrors those used for GPU programming: the threads (work-items) of instructions are grouped into groups (work-groups), which form a logically three-dimensional grid. The optimum choices of the number of work-groups, and the number of work-items per work-group dependent on the algorithm and the characteristics of the device in use.

The nodal DG scheme is naturally decomposed into three computational tasks: the computation of the first right-hand side terms of equations (7)-(8) (volume terms), the computation of the second ones (surface terms), and the update of discrete unknowns with the time stepping scheme. The computation of volume and surface terms mainly consists of matrix-vector products, while the update of unknowns is a purely local sum. This partition of the computation into separate kernels allows us to optimize each task, improving both load balancing and data transfers, considering the characteristics of both the task and the device. Thereby, the time-stepping loop iterations are performed by the main process, which calls the different kernels to execute operations on the device. The implementation of each kernel is based on the work of Klöckner et al. (2009). We note that a supplementary kernel allows us to generate signals at source points.



(a) (b)

Figure 1 *Representation of the half part of the computational domain with the coarsest mesh* (a) *and snapshot of the pressure at one instant* (b)*. The mesh cells are smaller in the upper medium (light gray) than in the lower one (dark gray). The source position (upper point) and the receiver position (lower point) are represented on figure* (a)*.*

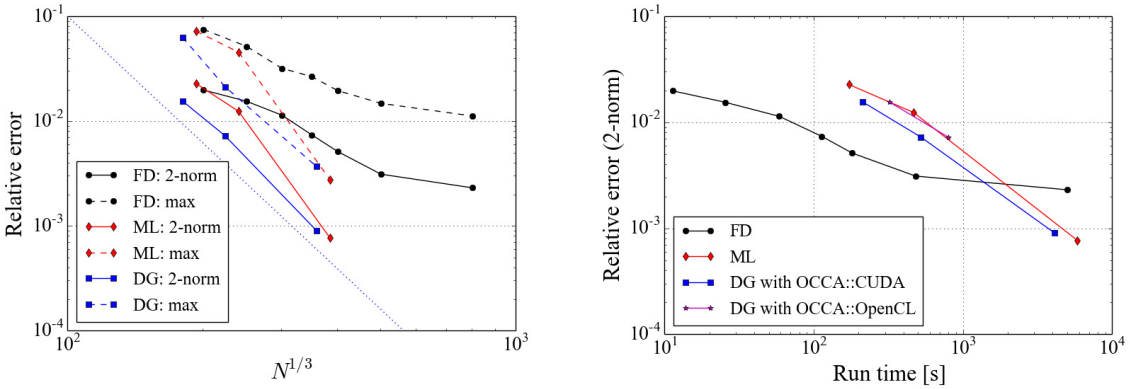
Results

In order to validate our implementation, we consider a benchmark used by Zhebel et al. (2014). In that work, a finite difference (FD) method and a mass-lumped finite element (ML) method were compared using OpenMP implementations. In this abstract, we present results of these implementations, obtained with the 16 cores of a dual Xeon (Sandy Bridge) 2.6 GHz E5-2670. For the DG runs, we use a single GPU NVIDIA Tesla K40 with both CUDA and OpenCL programming platforms through the OCCA framework.

The computational domain of the benchmark is a cube made of two media, which are separated with a dipping plane interface, as shown in Figure 1a. The length of each edge of the cube is 2km. The density is $1\mathrm{g/cm}^3$ in both media, and the velocity is $3\mathrm{km/s}$ in the upper one and $1.5\mathrm{km/s}$ in the other. A Ricker pulse is generated at a point source located 350m above the interface, and generate a spherical wave (Figure 1b). The pressure then is recorded at a 300m−offset receiver situated 250m above the interface, and the signal is compared to an exact solution. More details about this benchmark are given by Minisini et al. (2012) and Zhebel et al. (2014).

Figure 2 shows curves of relative errors as function of the characteristic mesh size and the running time. According to Zhebel et al. (2014), we consider both the 2−norm error and the maximum error on the recorded signal, divided by the maximum absolute value of the exact signal. For the DG runs, we use the same numerical setting than for the ML simulations: meshes with 294 508, 567 071 and 2 320 289 tetrahedra, third-order polynomial basis functions, and a CFL number set to 0.3.

We recover the classical h^{p+1} convergence of the DG scheme (figure 2a), which validates our implementation. For a given number of nodes N, the error values with the DG scheme are slightly smaller than with the ML scheme, and are close for the finest mesh. The computational performance of DG+GPU and ML+CPU implementations are similar (figure 2b). For the DG+GPU one, the run time is 33% faster with CUDA than with OpenCL. The finest mesh did not compute with OpenCL due to memory limitations of the OpenCL platform. Finally, the DG and ML schemes require both a smaller number of nodes and a smaller run time than the FD scheme to get the most accurate results.



(a) (b)

Figure 2 *Relative errors as a function of the characteristic mesh size* (a) *and the running time* (b). *Results of FD and ML methods obtained with OpenMP and a dual Xeon Processor. DG results are obtained using a GPU with both CUDA and OpenCL through OCCA. In the figure* (a)*, N is the total number of nodes, and the dotted line corresponds to the* 4^{th} *order of convergence.*

Conclusions

We have presented and validated a novel high-performance computational implementation for seismic wave propagation simulations. This implementation can be run on several hardware architectures thanks to the unified programming framework OCCA. In this abstract, we proposed preliminary results obtained with a single GPU and both OCCA::CUDA and OCCA::OpenCL. The extension of this implementation to more realistic cases for computation with different accelerator-aided clusters will be facilitated thanks to the OCCA framework, and will be presented in future works.

Acknowledgements

AM is Honorary BAEF Fellow and acknowledges Wallonie-Bruxelles International for an excellence grant. TW acknowledges generous support from the Shell Oil Company. Approved for release under RTI 95514115.

References

Hesthaven, J.S. and Warburton, T. [2007] *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, vol. 54. Springer.

Klöckner, A., Warburton, T., Bridge, J. and Hesthaven, J.S. [2009] Nodal discontinuous Galerkin methods on graphics processors. *Journal of Computational Physics*, **228**(21), 7863–7882.

Medina, D., St.-Cyr, A. and Warburton, T. [2014] OCCA: A unified approach to multi-threading languages. ArXiv/abs/1403.0968.

Minisini, S., Zhebel, E., Kononov, A. and Mulder, W.A. [2012] Efficiency comparison for continuous mass-lumped and discontinuous Galerkin finite-elements for 3D wave propagation. *Proceedings of the 74th EAGE Conference & Exhibition*, A004.

Mu, D., Chen, P. and Wang, L. [2013] Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using the graphic processing unit (GPU)-single-GPU implementation. *Computers & Geosciences*, **51**(0), 282 – 292.

Zhebel, E., Minisini, S., Kononov, A. and Mulder, W.A. [2014] A comparison of continuous mass-lumped finite elements with finite differences for 3-D wave propagation. *Geophysical Prospecting*, **62**(5), 1111–1125.