72nd EAGE Conference & Exhibition incorporating SPE EUROPEC 2010 Barcelona, Spain

Please read the information and the template instructions carefully before completing this call for papers!

The call for papers deadline is 20 January 2010.

Submission summary

Corresponding author

Name	Prof. Dr D. Komatitsch
Company	University of Pau, CNRS and INRIA, France
Department	INRIA IPRA MIGP
Address	Batiment IPRA MIGP Universite de Pau Avenue de l'Universite BP 1155 64013 Pau FRANCE

Extended abstract

Running 3D finite-difference or spectral-element wave propagation codes 25x to 50x faster using a GPU cluster

D. Komatitsch* (University of Pau, CNRS and INRIA, France), D. Michéa (BRGM, France), G. Erlebacher (Florida State University, Tallahassee, USA) & D. Göddeke (TU Dortmund, Germany)

Summary (Edit)

We first accelerate a three-dimensional finite-difference in the time domain (FDTD) wave propagation code by a factor of 50 using Graphics Processing Unit (GPU) computing on a NVIDIA graphics card with the CUDA programming language in the case of the fully heterogeneous elastic wave equation. We also implement Convolution Perfectly Matched Layers (CPMLs) on the graphics card to efficiently absorb outgoing waves on the fictitious edges of the grid. The methodology can be used for Maxwell's equations as well because their form is similar to that of the seismic wave equation written in velocity vector and stress tensor. We then implement a high-order spectral-element seismic wave propagation application on a cluster of GPUs. We compare it to a C+MPI implementation on a classical CPU cluster. We use mesh coloring to efficiently handle summation operations over degrees of freedom on an unstructured mesh. Using non-blocking MPI communications allows us to overlap the communications across the network and data transfers between the GPUs and the CPUs with calculations on the GPUs. We validate the CUDA and MPI implementation and analyze performance measurements. In average we obtain a speedup of 20x to 25x.

Extended abstract

File uploaded.
Original name: 'abstract_Komatitsch_EAGE_2010_Barcelona.pdf'
Size: 172kb

Introduction

Finite-difference (FD) techniques in the time domain (FDTD) are widely used to solve wave equations such as Maxwell's equations or the seismic wave equation. For a recent thorough review on FD applied to the seismic wave equation see e.g. Moczo et al. (2007). When more geometrical flexibility is needed for instance to handle geometrically complex models other techniques such as a spectral-element method (e.g., Liu et al. (2004); Chaljub et al. (2007); Tromp et al. (2008)) are often needed.

In recent years, computing on graphics cards (also known as 'Graphics Processing Unit (GPU) computing') has been used to accelerate non-graphical applications with respect to calculations performed on a classical Central Processing Unit (CPU), i.e., a processor core. GPU programming on NVIDIA graphics cards has become significantly easier with the introduction of CUDA, which is relatively easy to learn because its syntax is similar to C. Regarding FD, several applications have been ported to GPUs as early as 2004 (Krakiwsky et al., 2004; Micikevicius, 2009; Abdelkhalek et al., 2009; Michéa and Komatitsch, 2010). The spectral-element method has been successfully ported to GPUs by Komatitsch et al. (2009) and Komatitsch et al. (2010). Regarding FD for seismic reverse time migration in the case of an acoustic medium with constant density, Abdelkhalek (2007) and Micikevicius (2009) have recently introduced optimized implementations. Abdelkhalek et al. (2009) extended it to the acoustic case with heterogeneous density. Here we use CUDA to solve the seismic wave equation in the more complex fully heterogeneous (including for density) elastic case. We explain how we ported our finite-difference and our spectral-element seismic wave propagation codes to GPU graphics cards using CUDA.

The seismic wave equation and a classical finite-difference discretization

We consider a linear isotropic elastic rheology for the solid medium, and therefore the seismic wave equation can be written in the strong, i.e., differential, form

$$\begin{aligned} \rho \ddot{\mathbf{u}} &= \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \mathbf{f} \,, \\ \boldsymbol{\sigma} &= \mathbf{c} : \boldsymbol{\varepsilon} \,, \\ \boldsymbol{\varepsilon} &= \tfrac{1}{2}[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^T] \,, \end{aligned} \tag{1}$$

where $\mathbf{u}$ denotes the displacement vector, $\boldsymbol{\sigma}$ the symmetric, second-order stress tensor, $\boldsymbol{\varepsilon}$ the symmetric, second-order strain tensor, $\mathbf{c}$ the fourth-order stiffness tensor, ρ the density, and $\mathbf{f}$ an external source force. The double tensor contraction operation is denoted by a colon, a superscript T denotes the transpose, and a dot over a symbol indicates time differentiation. The material properties of the solid, $\mathbf{C}$ and ρ, can be spatially heterogeneous. In the classical velocity-stress formulation that is used in most FD implementations, one rewrites Eq. (1) as a first-order system whose unknowns are the velocity vector $\mathbf{v}$ and the stress tensor $\boldsymbol{\sigma}$. The boundary condition at the free surface of the medium is that the traction vector $\boldsymbol{\tau}$ must be zero everywhere at the surface.

Discretization of the first-order system is classically performed based on a staggered grid. Partial spatial derivatives are approximated by discrete operators involving differences between adjacent grid points. Time evolution is performed based on a staggered central finite-difference approximation. In the oil industry as well as in seismology in most cases one is interested in simulating a semi-infinite medium with a free upper surface. All the edges of the grid except the top edge are then artificial and outgoing waves should be absorbed there in order to simulate a semi-infinite medium. We use the unsplit CPML technique of Komatitsch and Martin (2007).

Our FD elastic wave propagation algorithm on a graphics card using CUDA

The main difficulty when implementing a finite-difference code on a GPU comes from the computation stencil used. For a fourth-order spatial operator, the thread that handles the calculation of point (i, j, k) needs to know the fields (and therefore access the arrays) at 12 neighboring points. This implies that 13 accesses to global memory are needed on the GPU to handle each grid point, which is a very high value,

keeping in mind that access to global GPU memory is very slow. But because threads that belong to the same block of threads can access common values using much faster GPU shared memory, we can significantly reduce this number of memory accesses per grid point and thus drastically improve performance. Because we use an explicit time scheme, values computed in the whole grid at a given time t depend only on past values already computed and are therefore independent. We can thus implement spatial parallelism by computing many grid points in parallel. Because GPUs require massive multithreading based on very lightweight threads, we use a different thread to handle each grid point.

CUDA threads are grouped in thread blocks. The most intuitive approach is to use a cubic distribution of threads. This way, each thread will load the values of the arrays at the grid point it handles from global memory to shared memory, and thus inside a given block of threads many neighboring points of the finite-difference stencil are automatically loaded to shared memory by the other threads of the block because with such a cubic distribution of threads by definition they handle these neighboring points. But it is more efficient to turn to a 2D approach introduced by Micikevicius (2009), which uses a sliding computation window. The 2D approach consists in tiling a 2D cut plane of the volume (for instance in the X and Y directions of the finite-difference grid) with 2D tiles, each tile corresponding to a block of threads. One can then iterate along the third (and last) direction, e.g. the Z direction, shifting the halo points for this last direction in registers organized in a pipeline fashion, taking advantage of the fact that access to these registers is extremely fast.

Our experimental setup is composed of an NVIDIA GeForce 8800 GTX video card with 768 MB of memory installed in the PCI Express 1 bus of a quad-processor dual-core 64-bit AMD Opteron 2.8 GHz PC. We use a 3D model of size 38300 m $\times$ 32700 m $\times$ 5100 m discretized using a grid of 384 $\times$ 328 $\times$ 52 points, i.e., with grid cells of size 100 m in the three spatial directions, and CPML layers having a thickness of 10 grid points. We validated our GPU implementation by comparing the results (not shown here) to an existing and already validated (Aochi and Douglas, 2006) serial implementation in C for a classical processor core. The agreement obtained was excellent.

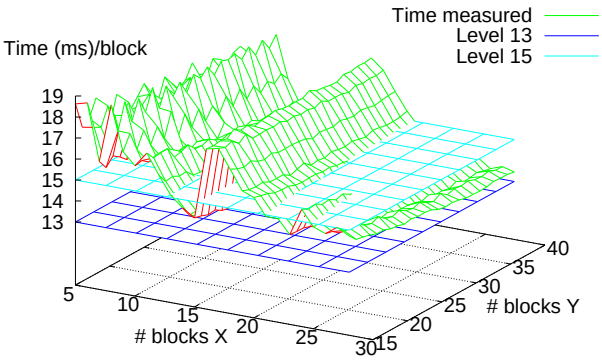

Figure 1 *Scaling of the GPU three-dimensional finite-difference code as a function of the number of thread blocks along the X and Y axes of the grid. The size of each thread block and thus the number of calculations that it performs does not vary, therefore in an ideal case we should get a flat surface. Scaling along X suffers from moderate variations that are likely due to undocumented hardware requirements. The blue and cyan horizontal planes indicate reference levels for comparison.*

Let us study the performance of the GPU code and compare it to the performance of the CPU code. The total elapsed time to perform the simulation (being the only user on the machine) is 7813 s for the CPU code (running on one CPU core) and 177 s for the GPU code. This means that the so-called 'speedup', defined as the ratio between the time spent running the whole simulation on the CPU and the time spent running the same simulation on the GPU, is 7813 / 177 = 44. For thicker, i.e., more accurate CPML layers having a thickness of 16 grid points, because multiples of 16 give significantly more efficient memory accesses on the GPU, the speedup is higher and reaches 55. In Figure 1 we represent the scaling of the code, i.e., the variation of the time spent by each block of threads to compute its share of the calculations on the GPU when the number of thread blocks varies along the X and Y axes of the grid. To do so, we make the total size of the model vary along the X and Y axes but keep a constant size for each block of threads, and we also use only model sizes that are multiples of the size of the basic thread blocks in order to keep all threads active. We therefore express the total size of each grid axis equivalently as a number of thread blocks. Since the size of each thread block and thus the number of

calculations that it performs does not vary, in an ideal case we should get a flat surface. But in practice performance per block is higher when we use more blocks, i.e., a larger finite-difference grid, because the scheduler of the GPU graphics card then has more opportunities to overlap the latency of accesses of blocks to global memory by calculations performed by other blocks that are ready to start computing (i.e., that are already done accessing global memory). However, despite the coalesced memory accesses that we implemented, scaling along X is not regular and suffers from moderate variations that are likely due to undocumented hardware requirements.

Our spectral-element elastic wave propagation algorithm on a cluster of 192 GPUs

As mentioned above, in cases that require more geometrical flexibility we often resort to the Spectral Element Method (SEM) to simulate numerically the propagation of seismic waves resulting from active seismic acquisition experiments in the oil industry or from earthquakes in the Earth. The SEM is a high-order finite-element method that solves the variational form of the elastic wave equation in the time domain on a non-structured mesh of elements, called spectral elements, in order to compute the displacement vector of any point of the medium under study. To represent the displacement field in an element, the SEM uses Lagrange polynomials of degree 4 to 10, typically, for the interpolation of functions. These Lagrange polynomials are defined in terms of control points that are chosen to be the Gauss-Lobatto-Legendre (GLL) points because they lead to an exactly diagonal mass matrix, i.e., no resolution of a large linear system is needed.

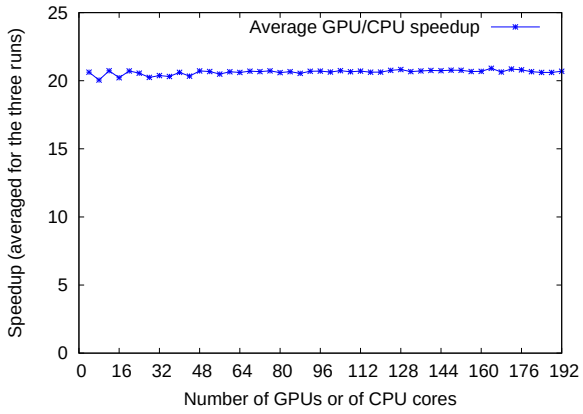

Figure 2 *GPU/CPU speedup obtained for our spectral-element seismic wave modeling application, called SPECFEM3D, on a large GPU cluster. We use 90% of the 4 GB of memory of each GPU, the rest being left for the system. The average speedup for the 48 measurements is 20.63, the maximum is 20.91 and the minimum is 20.05. The SPECFEM3D package is available open source under the GNU GPL version 2 license from* `www.geodynamics.org`.

Our goal is to port this application to a large cluster of GPUs, using the Message-Passing Interface (MPI) between compute nodes to exchange information between the GPUs. There are several challenges to address in mapping SEM computations to a GPU cluster. The elements that compose the mesh slices are in contact through a common face, edge or point. To allow for overlap of communication between cluster nodes with calculations on the GPUs, we create – inside each slice – a list of all these 'outer' elements, and an analogous list of the 'inner' elements. We compute the outer elements first, as it is done classically. Once these computations have been completed, we copy the associated data to the respective MPI buffers and issue a non-blocking MPI call, which initiates the communication and returns immediately. While the messages are traveling across the interconnect, we compute the inner elements. Achieving effective overlap requires that the ratio of the number of inner to outer elements be sufficiently large, which is the case for large enough mesh slices. Under these conditions, the MPI data transfer will statistically likely complete before the completion of the computation of the inner elements. We note that to achieve effective overlap on a cluster of GPUs, this ratio must be larger than for classical CPU clusters, due to the speedup obtained by the GPUs. The elementwise contributions need to be assembled at each global point of the mesh. Each such point receives contributions from a varying number of elements, which calls for an atomic summation, i.e., an order-independent sequential accumulation. We decouple these dependencies, which do not parallelize in a straightforward manner, by using a mesh coloring scheme to create sets of independent elements in the mesh (Komatitsch et al., 2009). This results in one CUDA kernel call per color, and total parallelism inside each color.

The machine we use is a cluster of 48 Teslas S1070 at CCRT/ CEA/GENCI in Paris, France; each Tesla S1070 has four GT200 GPUs and two PCI Express-2 buses (i.e., two GPUs share a PCI Express-2 bus). The GT200 cards have 4 GB of memory. The Teslas are connected to BULL Novascale R422 E1 nodes with two quad-core Intel Xeon Nehalem processors. We use mesh slices of 446,080 spectral elements each. Each slice contains approximately 29.6 million unique grid points, i.e., 88.8 million degrees of freedom, corresponding to 3.6 GB (out of 4 GB) of memory used per GPU. The largest possible problem size, using all 192 GPUs in the cluster, thus has 17.05 billion unknowns. Figure 2 shows GPU/CPU speedup when using between 4 GPUs and the whole GPU cluster, i.e., 192 GPUs. The average speedup for the 48 measurements performed is 20.63, the maximum is 20.91 and the minimum is 20.05.

Conclusions

We have accelerated a three-dimensional finite-difference wave propagation code by a factor of about 50 using a cheap NVIDIA GPU graphics card and the CUDA programming language. We have simulated seismic wave propagation in the heterogeneous elastic case, using CPML absorbing layers on the fictitious edges of the grid and implementing the finite-difference parallelization technique for GPUs of Micikevicius (2009), with the additional use of texture fetching in CUDA to compensate for the lack of shared memory on the graphics card. Due to the fact that the seismic wave equation written in velocity vector and stress tensor has the same second-order linear hyperbolic form as Maxwell's equations written in $\mathbf{E}$ and $\mathbf{B}$ (or $\mathbf{H}$) vectors, our GPU implementation can also be applied to Maxwell's equations.

We have also implemented a high-order spectral-finite-element application on a large GPU-enhanced cluster. Mesh coloring has enabled an efficient summation at shared degrees of freedom in the assembly process over an unstructured mesh. We have used non-blocking MPI and shown that computations and communications over the network and between the CPUs and the GPUs are almost fully overlapped because the GPU solver scales excellently up to 192 GPUs. It achieves a speedup of 20x over a carefully tuned equivalent CPU code.

References

Abdelkhalek, R. [2007] *Évaluation des accélérateurs de calcul GPGPU pour la modélisation sismique*. Master's thesis, ENSEIRB, Bordeaux, France.

Abdelkhalek, R., Calandra, H., Coulaud, O., Roman, J. and Latu, G. [2009] Fast seismic modeling and reverse time migration on a GPU cluster. *High Performance Computing & Simulation 2009*, Leipzig, Germany, 36–44.

Aochi, H. and Douglas, J. [2006] Testing the validity of simulated strong ground motion from the dynamic rupture of a finite fault by using empirical equations. *Bull. Earthq. Eng.*, **4**(3), 211–229.

Chaljub, E., Komatitsch, D., Vilotte, J.P., Capdeville, Y., Valette, B. and Festa, G. [2007] Spectral element analysis in seismology. In: Wu, R.S. and Maupin, V. (Eds.) *Advances in wave propagation in heterogeneous media*. Elsevier - Academic Press, vol. 48 of *Advances in Geophysics*, 365–419.

Komatitsch, D., Erlebacher, G., Göddeke, D. and Michéa, D. [2010] High-order finite-element seismic wave propagation modeling with MPI on a large-scale GPU cluster. *J. Comput. Phys.*, submitted.

Komatitsch, D. and Martin, R. [2007] An unsplit convolutional Perfectly Matched Layer improved at grazing incidence for the seismic wave equation. *Geophysics*, **72**(5), SM155–SM167, doi:10.1190/1.2757586.

Komatitsch, D., Michéa, D. and Erlebacher, G. [2009] Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA. *Journal of Parallel and Distributed Computing*, **69**(5), 451–460, doi:10.1016/j.jpdc.2009.01.006.

Krakiwsky, S.E., Turner, L.E. and Okoniewski, M.M. [2004] Graphics processor unit (GPU) acceleration of a finite-difference time-domain (FDTD) algorithm. *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems*, 265–268.

Liu, Q., Polet, J., Komatitsch, D. and Tromp, J. [2004] Spectral-element moment tensor inversions for earthquakes in Southern California. *Bull. Seismol. Soc. Am.*, **94**(5), 1748–1761, doi:10.1785/012004038.

Michéa, D. and Komatitsch, D. [2010] Accelerating a 3D finite-difference wave propagation code by a factor of 50 using a graphics card. *Geophys. J. Int.*, in press.

Micikevicius, P. [2009] 3D finite-difference computation on GPUs using CUDA. *GPGPU-2: Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units*, Washington, DC, USA, 79–84.

Moczo, P., Robertsson, J. and Eisner, L. [2007] The finite-difference time-domain method for modeling of seismic wave propagation. In: Wu, R.S. and Maupin, V. (Eds.) *Advances in wave propagation in heterogeneous media*. Elsevier - Academic Press, vol. 48 of *Advances in Geophysics*, chap. 8, 421–516.

Tromp, J., Komatitsch, D. and Liu, Q. [2008] Spectral-element and adjoint methods in seismology. *Communications in Computational Physics*, **3**(1), 1–32.