Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition

Taro Okamoto[1], Hiroshi Takenaka[2], Takeshi Nakamura[3], and Takayuki Aoki[4]

[1]*Department of Earth and Planetary Sciences, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo 152-8551, Japan*
[2]*Department of Earth and Planetary Sciences, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan*
[3]*Earthquake and Tsunami Research Project for Disaster Prevention, Japan Agency for Marine-Earth Science and Technology, 3173-25, Showa-machi, Kanazawa-ku, Yokohama, Kanagawa 236-0001, Japan*
[4]*Global Scientific Information and Computing Center, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo 152-8550, Japan*

(Received September 16, 2010; Revised November 21, 2010; Accepted November 21, 2010; Online published February 3, 2011)

We adopted the GPU (graphics processing unit) to accelerate the large-scale finite-difference simulation of seismic wave propagation. The simulation can benefit from the high-memory bandwidth of GPU because it is a "memory intensive" problem. In a single-GPU case we achieved a performance of about 56 GFlops, which was about 45-fold faster than that achieved by a single core of the host central processing unit (CPU). We confirmed that the optimized use of fast shared memory and registers were essential for performance. In the multi-GPU case with three-dimensional domain decomposition, the non-contiguous memory alignment in the ghost zones was found to impose quite long time in data transfer between GPU and the host node. This problem was solved by using contiguous memory buffers for ghost zones. We achieved a performance of about 2.2 TFlops by using 120 GPUs and 330 GB of total memory: nearly (or more than) 2200 cores of host CPUs would be required to achieve the same performance. The weak scaling was nearly proportional to the number of GPUs. We therefore conclude that GPU computing for large-scale simulation of seismic wave propagation is a promising approach as a faster simulation is possible with reduced computational resources compared to CPUs.

Key words: Seismic wave propagation, finite-difference method, GPU, parallel computing, three-dimensional domain decomposition.

1. Introduction

Simulation of seismic wave propagation is essential in modern seismology: the effects of irregular topography, irregular internal discontinuities, and internal heterogeneity on the seismic waveforms must be precisely modeled in order to probe the Earth's and other planets' interiors, to study earthquake sources, and to evaluate the strong ground motions due to earthquakes. Developing methods for large-scale, high-performance simulation is important because in real applications more than one billion grid points are required (e.g., Olsen *et al*., 2008; Furumura, 2009).

The GPU (graphics processing unit) is a remarkable device due to its multi-core architectures and high memory bandwidth (Fig. 1). The GPU delivers extremely high computing performance at a reduced power and cost compared to conventional central processing units (CPUs). Simulation of seismic wave propagation is a *memory intensive* problem which involves a large amount of data transfer between the memory and the arithmetic units, while the number of arithmetic computations is relatively small. Thus, the simulation can benefit from the high-memory bandwidth of the GPU, and several approaches to adopt GPU to the simulation have been proposed recently (e.g., Abdelkhalek *et al*., 2009; Aoi *et al*., 2009; Komatitsch *et al*., 2009, 2010; Micikevicius, 2009; Okamoto *et al*., 2009; Michéa and Komatitsch, 2010).

We describe here our approach to adopt GPU computing to the large-scale simulation of seismic wave propagation based on the finite-difference method (FDM). First, we discuss the implementation of the core part of the simulation for the single-GPU case. Second, we discuss the multi-GPU case in order to treat large-scale problems.

2. Single-GPU Case

We apply the time-domain, staggered-grid, three-dimensional (3D) finite-difference scheme (e.g., Graves, 1996). The field variables are particle velocity (v_i: three components) and stress (τ_{ij}: six components). Assuming isotropic and elastic material, we need three material parameters (Lamé coefficients and density). Thus, twelve variables are assigned to a unit cell (i.e., a heterogeneous formulation: Fig. 2(a)). The precision of finite-difference is fourth-order in space and second-order in time. We apply a free-surface condition at the top boundary, the absorbing boundary condition (Cerjan *et al*., 1985) near the side and the bottom boundaries, and the A1 absorbing condition (Clayton and Engquist, 1977) at the bottom. A periodic condition is imposed on the side boundaries. A homogeneous half-space and an isotropic source are used for all calculations performed and reported in this paper.

We use the TSUBAME-1.2 grid cluster in the Global Sci-

Copyright © The Society of Geomagnetism and Earth, Planetary and Space Sciences (SGEPSS); The Seismological Society of Japan; The Volcanological Society of Japan; The Geodetic Society of Japan; The Japanese Society for Planetary Sciences; TERRAPUB.

doi:10.5047/eps.2010.11.009

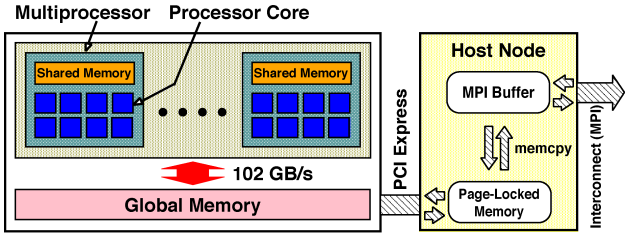

Fig. 1. A simplified diagram of a GPU used in this study. A GPU has 30 multiprocessors, and each multiprocessor has eight processor cores (240 cores in total). The shared memory can be accessed from every core in a multiprocessor, while registers are local to each core. Several tens of registers are typically available for a thread. Latency of the shared memory is two orders of magnitude lower than that of global memory (Abdelkhalek *et al.*, 2009; Micikevicius, 2009).

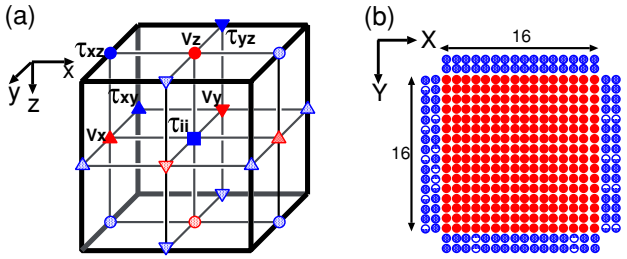

Fig. 2. (a) Schematic illustration of the unit cell. (b) Example of 2D block assigned to the shared memory. The blocksize is 16 × 16 (red). For fourth order finite-difference, two ghost points are also required to be assigned (blue).

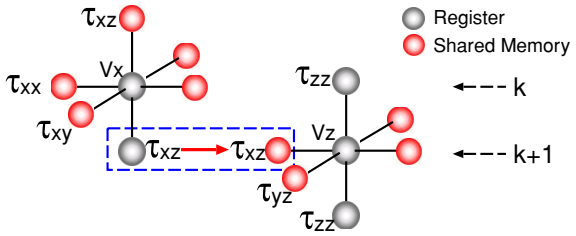

Fig. 3. Example of the use of the shared memory and registers. Values on red points are stored in shared memory and those on black points in registers. In the computations for k-th depth, the stress component τ_{xz} is stored in the register. For the next depth, the value in the register is moved to the shared memory to reduce the amount of data transfer from the global memory.

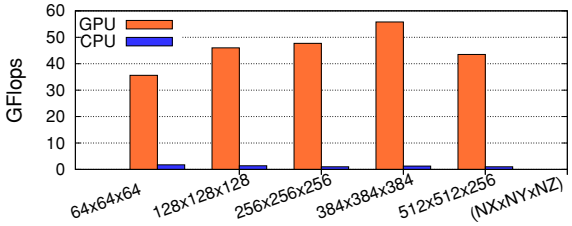

Fig. 4. Performance comparison between the single GPU and single-core of host CPU. The triplet values denote the number of unit cells.

entific Information and Computing Center, Tokyo Institute of Technology. The processors of the host nodes are eight dual-core AMD Opteron 880 (16 cores per 1 node, 2.4 GHz) and the interconnect is Infiniband (10 Gbps). The GPUs installed in the cluster are NVIDIA Tesla S1070s (1.44 GHz): each S1070 has four GPUs, and each host node is connected to two GPUs in a S1070 via a PCI-Express 1.0×8 (i.e., two GPUs per 1 node). We use NVIDIA CUDA C for the GPU programs. For the FDM program on the host CPU, we use PGI Fortran. Single-precision arithmetic is used in all of the computations on both the GPU and CPU. The theoretical peak single-precision performances are 9.6 GFlops for a single core of the host CPU and 1036 GFlops for a single GPU in the S1070 unit: the GPU is 108-fold faster in terms of computation time than a single core of the host CPU. The memory bandwidths are 5.4 GB/s (gigabyte/s) for a single host CPU and 102 GB/s for a single GPU: GPU is 19-fold faster in bandwidth than the host CPU.

In GPU computing, all data are stored in the *global memory* (Fig. 1). As described above, the bandwidth of the global memory is much higher than that of CPUs (e.g., 25.6 GB/s in the case of Intel Core i7). However, 400–600 clock cycles of memory latency still occurs in transferring the data between the global memory and the multiprocessors. Thus, we must use the fast (but small) memories in the multiprocessors, i.e., the *registers* and the *shared memory*, as software-managed cache memories to reduce the amount of data transfer from the global memory. That is, we explicitly copy the data in a small part (a *block*) of the FDM domain from the global memory to the shared memory and registers, and we reuse the data stored in the shared memory and registers (e.g., Aoki, 2009). Also, the transfer rate is better for grouped memory transaction using blocks of the proper size than that for serialized, one-by-one memory transaction.

We assign a 2D block in the shared memory to store the variables on a XY-plane (Fig. 2(b)), because the size of the shared memory (16 kB) is not sufficient for a 3D block. The variables which are not on the XY-plane are stored in the registers (Fig. 3). As the calculation loop proceeds to the next XY-plane, the data in the registers are moved to the shared memory and vice versa: for this movement, no data access to the global memory occurs. The use of the shared memory and registers as described has been applied by Abdelkhalek *et al.* (2009) and Micikevicius (2009) for the acoustic case, and by Michéa and Komatitsch (2010) and Okamoto *et al.* (2010) for the elastic case. Also, we define the material parameters only at the center of the unit cell. The material parameters at the grid points for particle velocities and shear stresses are computed at every time step to reduce the access to the global memory. In computing the material parameters, we apply a method proposed by Takenaka *et al.* (2009), as this method allows land topography and irregular fluid-solid interface (e.g., ocean bottom) to be incorporated in a unified manner.

The performance of the single-GPU finite-difference code is shown in Fig. 4. For Flops, we counted the number of floating point operations in the code, including those for the computations of the material parameters, and divided the number of operations by the time required for the time-step loop. For the fourth-order finite-difference, the Flop counts of the GPU and CPU programs are 185 and 176 per 1 unit cell, respectively (note that we omit the absorbing condition (9 Flops count) in the CPU program). With the GPU

program, a maximum performance of 56 GFlops (3.3 ms per 10^6 unit cells per time step) was achieved for a FDM domain with a size of 384 × 384 × 384 (about 2.7 GB of memory). This was about 45-fold faster than that achieved by the CPU program (1.2 GFlops for the same FDM domain). Thus, we confirm that GPU can accelerate the simulation of finite-difference seismic wave propagation.

In the examples above, we have fixed the blocksize to 16 × 16 (Fig. 2(b)). When we reduced the blocksize to 2 × 2, the performance markedly decreased to 3.5 GFlops for the FDM domain of 384 × 384 × 384. The reason for this is that, for small blocksize, the access to the global memory increases and the memory transfer rate decreases. Thus, the optimized use of the fast memories is essential for performance.

3. Multi-GPU Case

We divide the FDM domain into *subdomains* and allocate computation for a single subdomain to a single GPU by using the MPI library. We adopt *3D* domain decomposition (Fig. 5) because the communication time decreases with decreasing size of the subdomain: the communication time is proportional to the surface area of the subdomain, and the surface area decreases with the size of the subdomain. On the other hand, with *1D* domain decomposition (e.g., Micikevicius, 2009), the domain can be extended only along one direction, and the communication time does not decrease with the size (or the thickness) of the subdomain because of the fixed size of the ghost zones (Fig. 5).

It is possible to extend the memory array of the internal grid points to cover the ghost zones on the side faces of the subdomain (Fig. 6(a)). However, we found that it took quite a long time to separately send the resultant non-contiguous data from GPU to the host node by repeated calls of memory transfer function. (Note that direct communication between GPUs is not available. The data must be sent from GPU to the host node in order that the data be exchanged with the other nodes (Fig. 1).) Thus, we prepare contiguous memory buffers for ghost zones (Fig. 6(b)) so that we are able to copy the data from GPU to the host node by a single call of memory transfer function. For the outermost blocks we copy the data in the memory buffer to the ghost points in the shared memory (Fig. 2(b)) and vice-versa.

Also, we overlap the communication and computation by using the asynchronous GPU–host data transfer function and non-blocking MPI functions (e.g., Abdelkhalek *et al.*, 2009; Aoki, 2010; Ogawa *et al.*, 2010): the former function is used for data transfer between GPU and the host concurrent with the computations in GPU, and the latter functions for data transfer between the nodes concurrent with the computations. Since we adopt the 3D domain decomposition, we first compute for the ghost points—not only on the top and bottom but also on the sides: 16 × 16 points (Fig. 2(b)) in the outermost blocks on the sides are processed. Second we start the computation for the remaining internal grid points and the communication procedures simultaneously. For the asynchronous data transfer between GPU and host, we must use the page-locked host memory (Fig. 1) which is not compatible with the MPI library on TSUBAME-1.2. Thus, we further copy the data to a (usual) memory buffer ("memcpy" in Fig. 1): this results in additional time on TSUBAME-1.2.

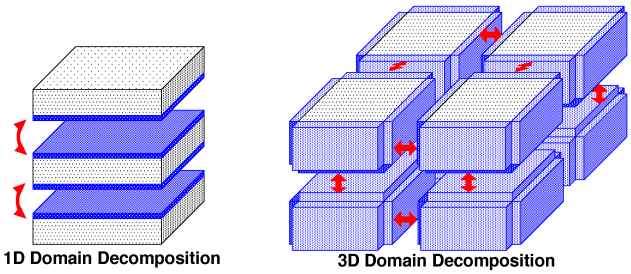

Fig. 5. Schematic illustration of 1D and 3D decomposition. Blue region indicates the ghost zones.

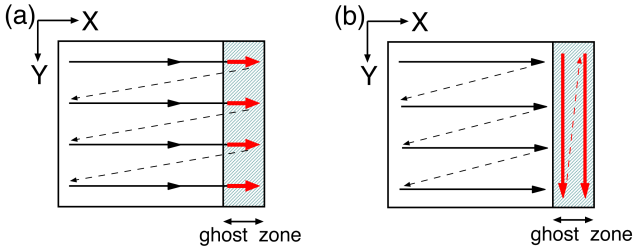

Fig. 6. Schematic illustration of ghost zone of a subdomain. Arrows denote the alignment of memory. (a) Non-contiguous memory alignment in the ghost zone (red arrows). The memory array of the internal points is extended to cover the ghost points. (b) Contiguous memory alignment. A memory buffer separate from that of the internal domain is used.

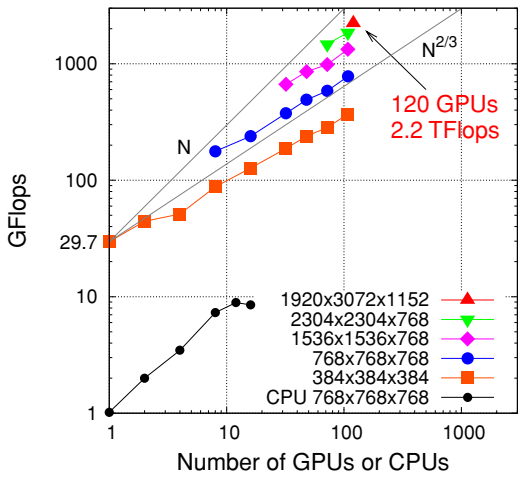

Fig. 7. Scalability of multi-GPU computing with 3D decomposition. Each triplet value shows the size of the FDM domain. The complete scalability (proportional to the number of GPUs, N) and a scalability proportional to $N^{2/3}$ are denoted by gray lines, respectively. For GPU, we used CUDA C and MPI library. For CPU, we parallelized the single-CPU Fortran code by using OpenMP and executed the program on a single host node.

The performance of our multi-GPU code thus programmed is shown in Fig. 7. The largest subdomain size was 384 × 384 × 384 in each case (e.g., 2 × 2 × 2 (= 8) subdomains were used in the case of 768 × 768 × 768 with 8 GPUs). Obviously, the performance increased with the number of GPUs. That is, the program was effectively parallelized. We achieved a performance of 2.2 TFlops by us-

Table 1. Part-wise processing time (s) of the multi-GPU program.

	Computation		Communication		
# GPUs	Side points	Inner points	GPU–host	Memcpy	MPI
1	35.8	34.2	—	—	—
4	13.8	7.4	10.1	10.1	8.1
8	7.0	3.7	6.5	6.9	5.2

FDM domain size: 384 × 384 × 384. Time steps: 200.

ing 120 GPUs for a domain size of 1920 × 3072 × 1152 with about 330 GB of total memory (0.083 ms per 10^6 unit cells per time step).

The *weak scaling*, defined for a fixed subdomain size, is observed as the variation in performance in the cases with smallest number of GPUs for each domain size. It was nearly proportional to N, i.e., near ideal (Fig. 7).

In Fig. 7, we also compared the performance of the GPU and CPU programs. The performance of the CPU program scaled with the number of cores up to eight but not beyond that (partly because we used OpenMP and 1D decomposition for the CPU program). Even if a complete scalability beyond eight cores were to be assumed, nearly 2200 cores of CPU would be required to obtain the same performance as that achieved by 120 GPUs.

Based on the above results, we conclude that GPU computing for the large-scale simulation of seismic wave propagation is a promising approach: faster simulation is possible at reduced computational resources compared to CPUs.

4. Discussion

The *strong scaling*, defined as the performance variation for a fixed total FDM domain size, was not proportional to the number of GPUs, N, but appeared to be proportional to $N^{2/3}$. This (non-ideal) scalability was the result of long communication and computation times for the ghost points because the number of the ghost points was proportional to the surface area of the subdomain, and the surface area was proportional to $N^{-2/3}$. Indeed, we measured the time for processing the side points (including ghost points), internal points, and the communications separately and determined that the communication time was the longest (Table 1).

As already pointed out by Aoki (2010) and Ogawa *et al.* (2010), the time for copying the data between the page-locked memory and the MPI buffer ("memcpy" time) was about one-third of the total communication time. As a result, in the above cases, the "memcpy" time was longer than the computation time for internal points. Nevertheless, the overlapping method is important as the technology for eliminating the "memcpy" procedure has recently been released. Peripheral bus and interconnect faster than those adopted by TSUBAME-1.2 cluster are now also available.

We also found that the time for side points was long: the memory mapping procedure between the ghost points in the shared memory and the contiguous memory buffer took a long time (Table 1). As a result, single-GPU performance fell by about 47% (from 55.8 GFlops for the single-GPU program to 29.7 GFlops for the multi-GPU program) due to the processing time for ghost point mapping. Further optimization is necessary for the mapping procedure.

In real applications, not only performance but also accuracy is important. We compared the waveforms computed by multi-GPUs (GPU-waveforms) and multi-CPUs (CPU-waveforms) for a case of 512 × 512 × 256 and 1500 time steps (7.5 s). The root mean square residuals between the CPU- and GPU-waveforms normalized by the RMS amplitude of corresponding CPU-waveforms were $(2\text{–}9)\times10^{-6}$. Thus, we confirmed that both waveforms were almost identical.

Acknowledgments. The course on GPU computing held at Global Scientific Information and Computing Center, Tokyo Institute of Technology was quite helpful. We are grateful to Tsugunobu Nagai for supporting this research. Comments made by two anonymous reviewers were very helpful in improving the manuscript.

References

Abdelkhalek, R., H. Calandra, O. Coulaud, J. Roman, and G. Latu, Fast seismic modeling and reverse time migration on a GPU cluster, *International Conference on High Performance Computing & Simulation*, 36–43, 2009.

Aoi, S., N. Nishizawa, and T. Aoki, 3-D wave propagation simulation using GPGPU, *Programme and Abstracts, Seismol. Soc. Jpn., 2009 Fall Meeting*, abstract A12–09, 2009.

Aoki, T., Full-GPU CFD applications, *IPSJ Mag.*, **50**(2), 107–115, 2009.

Aoki, T., Multi-GPU Scalabilities for Mesh-based HPC Applications, *SIAM Conf. Parallel Processing for Scientific Computing (PP10)*, Seattle, Washington, February 26, 2010.

Cerjan, C., D. Kosloff, R. Kosloff, and M. Reshef, A nonreflecting boundary conditions for discrete acoustic and elastic wave equations, *Geophysics*, **50**, 705–708, 1985.

Clayton, R. and B. Engquist, Absorbing boundary conditions for acoustic and elastic wave equations, *Bull. Seismol. Soc. Am.*, **67**, 1529–1540, 1977.

Furumura, T., Large-scale simulation of seismic wave propagation in 3D heterogeneous structure using the finite-difference method, *J. Seismol. Soc. Jpn. (Zisin)*, **61**, S83–S92, 2009.

Graves, R. W., Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences, *Bull. Seismol. Soc. Am.*, **86**, 1091–1106, 1996.

Komatitsch, D., D. Michéa, and G. Erlebacher, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, *J. Parallel Distrib. Comput.*, **69**, 451–460, 2009.

Komatitsch, D., G. Erlebacher, D. Göddek, and D. Michéa, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, *J. Comp. Phys.*, **229**, 7692–7714, 2010.

Michéa, D. and D. Komatitsch, Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards, *Geophys. J. Int.*, doi: 10.1111/j.1365-246X.2010.04616.x, 2010.

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

Ogawa, S., T. Aoki, and A. Yamanaka, Multi-GPU scalability of phase-field simulation for phase transition—5 TFlop/s performance on 40 GPUs, *Trans. IPS Japan, Advanced Computing Systems*, **3**, 67–75, 2010.

Okamoto, T., H. Takenaka, and T. Nakamura, Computation of Seismic Wave Propagation With GPGPU, *Programme and Abstracts, Seismol. Soc. Jpn., 2009 Fall Meeting*, abstract P3–22, 2009.

Okamoto, T., H. Takenaka, and T. Nakamura, Simulation of seismic wave propagation by GPU, *Proceedings of Symposium on Advanced Computing Systems and Infrastructures*, 141–142, 2010.

Olsen, K. B., S. M. Day, J. B. Minster, Y. Cui, A. Chourasia, D. Okaya, P. Maechling, and T. Jordan, TeraShake2: Spontaneous rupture simulations of Mw 7.7 Earthquakes on the Southern San Andreas Fault, *Bull. Seismol. Soc. Am.*, **98**, 1162–1185, 2008.

Takenaka, H., T. Nakamura, T. Okamoto, and Y. Kaneda, A unified approach implementing land and ocean-bottom topographies in the staggered-grid finite-difference method for seismic wave modeling, *Proc. 9th SEGJ Int. Symp.*, CD-ROM Paper No.37, 2009.

T. Okamoto (e-mail: okamoto.t.ad@m.titech.ac.jp), H. Takenaka, T. Nakamura, and T. Aoki