Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Computer Science 18 (2013) 1255 – 1264
Procedia Computer Science

2013 International Conference on Computational Science

Multi-GPU Implementation of a 3D Finite Difference Time Domain Earthquake Code on Heterogeneous Supercomputers

Jun Zhou[a,b,*], Yifeng Cui[a], Efecan Poyraz [a,b], Dong Ju Choi [a], Clark C. Guest [b]

[a] *San Diego Supercomputer Center, Univ. of California at San Diego, 10110 HopkinsDrive, La Jolla, CA92093, United States*
[b] *Dept. of Electrionic and Computer Engineering, Univ. of California at San Diego, 9500 Gilman Drive, La Jolla, CA92093, United States*

Abstract

We have developed a highly scalable 3D Finite Difference GPU code for use in earthquake engineering and disaster management through regional petascale earthquake simulations. This MPI-CUDA code is based on a widely-used wave propagation code called AWP-ODC and restructured for high throughput and efficiency on a heterogeneous computing architecture. We present an effective communication reduction technique for leveraging GPUs with minimal PCI-e overhead, and a novel overlapping method to fully hide data communication latency between GPUs. The optimization concept used in this work can be extended to general stencil computing on a structured grid. The benchmarks demonstrated sustained 100 TFlops in single precision for 49 billion mesh points using 952 GPUs on the NCCS Titan Phase 5 system, which is a 77-fold speedup compared to the CPU version of the code. This multi-GPU implementation has been validated and used for a large-scale verification wave propagation simulation of Mw5.4 Chino Hills earthquake using 128 GPUs.

Keywords: Earthquake Simulations; CUDA-MPI; Enhanced Overlapping Design; Heterogeneous Supercomputers.

1. Introduction

Anelastic Wave Propagation by Olsen, Day and Cui (AWP-ODC) is a 3D Finite Difference Time Domain (FDTD) earthquake simulation code [1].This real world code has been used in recent years by Southern California Earthquake Center (SCEC) for large-scale earthquake simulations [2]. Accelerating AWP-ODC wave propagation modeling using Graphics Processing Units (GPUs) can dramatically improve time-to-solution and energy efficiency for calculating California state-wide physics-based seismic hazard curves on petascale heterogeneous computing resources [3].

The graphics processing unit (GPU) has quickly become an industry standard as an integrated part of today's mainstream computing systems. A variety of scientific applications have adopted GPU-assisted computing and achieved notable speedups, such as Computational Fluid Dynamic [4], N-Body Simulation [5], Numerical Weather Prediction [6], etc. As of December 2012, two of the top 10 supercomputers (Titan and Tianhe-1A) in the world are GPU clusters, using NVIDIA Kepler and Fermi GPUs [7]. The Gordon Bell Prize winner at SC'11 by T. Shimokawabe et al. from Tokyo Institute of Technology demonstrated a multi-GPU implementation of phase-field simulation modeling, which achieved 1.017 PFlops in single precision using

*Corresponding author (Jun Zhou). Tel: +001-858-822-6178
Email Address: j4zhou@ucsd.edu

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.292

4,000 GPUs along with 16,000 CPUs on the TSUBAME 2.0 Supercomputer [8]. A number of previous research projects focused on acceleration of earthquake simulations using GPUs [9-18]. For example, Micikevicius [9] described parallelization of 3D finite difference (FD) seismic equations and extended his work to four NVIDIA Tesla 10-serial (T-10) GPUs; Abdelkhalek et al. [10] demonstrated 10x and 30x speedups with their 3D FD implementation for reverse time migration (RTM) and seismic application respectively on 8 T-10 GPUs; Komtitsch et al. [11-14] ported finite difference, finite element and spectral elements codes for 3D seismic simulation modeling to GPU clusters, and scaled the codes up to 120 T-10 GPUs with 20x speedup; Song et al. [15] accelerated the Support Operator Rupture Dynamic (SORD) code with 14x speedup on 64 T-10 GPUs; and Okamato et al. [16-18] reached 2.2 TFlops on 120 GPUs of the TSUBAME supercomputer.

Despite improvements in GPU programming driven by the increasing attention of the HPC community, there is still a significant burden in porting FDTD earthquake simulations to CPU-GPU heterogeneous supercomputers. Because of the relatively low communication bandwidth between the CPU and GPU on supercomputers, the performance of FDTD applications is often bounded by data communication between CPU and GPU. Hence, a lot of room remains for programmers to tune the performance of their applications for the latest CPU-GPU hybrid computing architectures.

This paper reports on continued development after [19], and introduces a multi-GPU implementation of AWP-ODC. Some details of tuning techniques are presented for CPU-GPU heterogeneous supercomputers. We emphasize algorithm-level MPI-CUDA data locality and in particular a novel effective overlapping algorithm developed for hiding the latency caused by data communication between GPUs. We then verify the GPU code by comparing the results of an Mw5.4 Chino Hills, CA earthquake simulation obtained using GPU code with the one obtained by CPU code. At the end of the paper, we provide scalability results, on both the NCCS Titan Phase 5 GPU system [20] and the XSEDE Keneland (KIDS) located at NICS [22].

2. Motivation and Background

The AWP-ODC numerical model is based on the 3D finite difference time domain method, solving a 3D velocity-stress wave equation on a staggered-grid. The computation kernels are based on 3D 13-point stencil computations [2]. In 2011, we developed a CUDA version of AWP-ODC code for running on a single NVIDIA Fermi GPU. The optimized code running on single NVIDIA Tesla M2090 GPU achieves 15x speedup in computation compared to the CPU code running on an AMD single Istanbul CPU socket (6 threads/cores) [19]. To run large earthquake simulations, however, the CUDA code must be extended to run on multi-GPU heterogeneous architectures to make the full use of GPU computing capability for high computation performance.

In this work we present results obtained on the NCCS Titan Phase 5 system and XSEDE Keeneland system. NCCS Titan is the Oak Ridge Leadership Computing Facility's (OLCF) next generation, Cray XK7 based heterogeneous system [20]. Titan, upgraded from the Jaguar system, [20], provides a peak theoretical performance of more than 20 Peta-Flops and currently ranks first among the 'Top 500' of supercomputers [7]. The results on Titan presented in this paper are based on the Phase 5 system, consisting of 960 nodes. Each computing node is equipped with one AMD 16 core Opteron^{TM} 6200 series processor and one NVIDIA Tesla X2090 Fermi GPU accelerator connected to the processor via PCI Express 2.0. The interconnection between nodes is Gemini, which is a 3-dimensional torus topology providing over 20GB/sec bandwidth per node [21]. The Georgia Tech Keeneland Initial Delivery System (KIDS) located at NICS has an architecture similar to the Titan Phase 5 system, including 120 computing nodes with two INTEL Westmere hex-core CPUs and three NVIDIA Tesla M2090 Fermi GPUs per node [22].

3. Multi-GPU Implementation

The full version of the GPU code includes parallel IO, computation, and communication, similar to the CPU code. The parallel IO implementation follows the same pattern as the CPU code in general [2], as all I/O

operations are processed by the CPUs. The implementation of computation kernels in CUDA was well optimized and discussed in [19]. We focus here on communication optimizations on heterogeneous supercomputers. Our CPU-GPU communication model has two primary objectives: 1) to reduce the data communication frequency between CPU and GPU inside the computing nodes; 2) to maximize the overlap time between computation and communication. Note that the communication model developed in this application is quite general, and can be applied to other similar large-scale 3D stencil computations to improve efficiency on CPU-GPU based supercomputers.

3.1 Two-layer 3D Domain Decomposition

In the AWP-ODC-CPU implementation, the 3D domain is partitioned into a number of sub-domains and each sub-domain is mapped to a single CPU core for computation. For decomposition on CPU-GPU based heterogeneous supercomputers, the first step is the same, i.e. the 3D domain is partitioned into a number of sub-domains, and each sub-domain is reserved for computation on a single GPU. The second step is to partition the sub-domain inside GPU for streaming multiprocessors (SM). Thus, we partition the domain twice, and call the whole decomposition process "two-layer 3D domain decomposition". For the first partitioning step, we choose 2D decomposition in the X and Y directions instead of 3D decomposition in all directions. Suppose that the 3D domain is represented as (NX, NY, NZ) and the decomposition topology is (PX, PY, 1), meaning that the 3D domain with NX*NY*NZ mesh points is partitioned into PX, PY and 1 pieces in X, Y and Z directions respectively. As shown in Figure 1, the sub-domain for each GPU becomes (nx, ny, NZ), where $nx=NX/PX$ and $ny=NY/PY$. In this case, the MPI communications each CPU does can be reduced from six directions to four directions (south, north, west and east). Each GPU takes over the entire computation in the Z direction. Larger NZ can improve GPU computation efficiency since the memory is managed in the fast Z direction. The two-layer 3D domain decomposition is X&Y partitioning for GPUs and Y&Z partitioning for GPU SMs, which will be discussed later.

3.2 Communication Reduction

The largest latency for the multi-GPU implementation is GPU to GPU communication between different computing nodes. A message between two GPUs passes through the high-speed network once and the PCI Express 2.0 bus twice. Based on the decomposition described in Figure 1, the MPI communications take place only between sub-domains (1: nx, 1: ny, 1: NZ) in X and Y directions. Hence we represent the 3D sub-domain with a 2D XY plane to show the communication reduction method in Figure 2. The characteristics of 13-point stencil computation require a two-layer ghost cells region to compute the whole sub-domain. This means for the sub-domain output (1: nx, 1: ny, 1: NZ), we need input (-1: nx+2, -1: ny+2, 1: NZ). To be more specific, the velocity sub-domain (1: nx, 1: ny, 1: NZ) update requires stress input (-1: nx+2, -1: ny+2, 1: NZ), and the stress sub-domain (1: nx, 1: ny, 1: NZ) update requires velocity input (-1: nx+2, -1: ny+2, 1: NZ) at each iteration. In other words, the communication patterns used in the CPU code requires 8 GPU to GPU communications, where four are for the velocity data swap for the two-layer ghost cells region in four directions (west, east, north and south), and the other four are for the stress data swap.

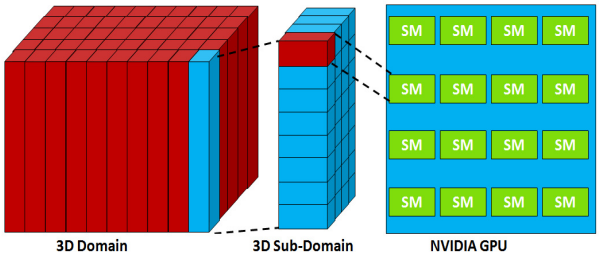

Fig.1. Process of two-layer 3D domain decomposition on CPU-GPU based heterogeneous supercomputers: first step X&Y decomposition for GPUs and second step Y&Z decomposition for GPU SMs.

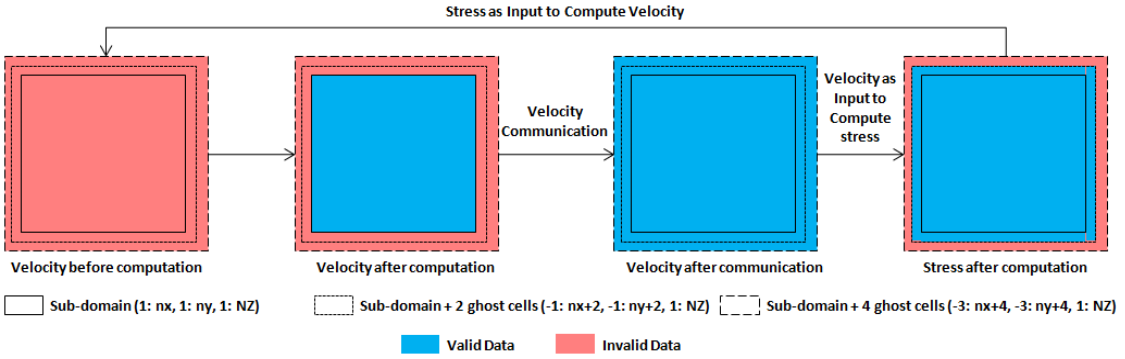

Fig.2. Communication Reduction Process: extend ghost cell region with extra 2-layers and utilize computation instead of communication to update the ghost cell region before stress computation. Here the 2D XY plane represents the 3D sub-domain. Note that no communication in Z direction is required, due to our 2D decomposition for GPUs.

Our enhanced communication reduction method extends the ghost cells region with an extra two-layers for velocity data (-3: nxt+4, -3: nyt+4, 1: NZ), though the target computation region is unchanged (1: nx, 1: ny, 1: NZ). The GPU to GPU communication for velocity of ghost cells, however, consists of data swapping of four layers instead of two, In this way the valid data region for velocity is (-3: nx+4, -3: ny+4, 1: NZ) after the communication. This means we now have sufficient velocity input (-3: nx+4, -3: ny+4, 1: NZ) to compute the stress for region (-1: nx+2, -1: ny+2, 1: NZ). This stress data region is computed and used for calculating the velocity for region (1: nx, 1: ny, 1: NZ) in the next iteration. In summary, only velocity data swapping is required in the new communication model, and the total number of GPU to GPU communications is thus reduced from eight to four per iteration. Adding to the fact that velocity has three components and stress has six, as illustrated in Table 1, the total size of GPU to GPU messages is now reduced by a factor of three. The significant reduction in communication costs using this new model helps leverage the GPU with minimal PCI-e overhead, allowing high scalability to large numbers of GPUs.

Table 1. GPU to GPU communication pattern comparison per iteration: before and after communication reduction.

Communication	Velocity		Stress	
	Frequency	Message Size	Frequency	Message Size
Before Communication Reduction	4	6*(nx+ny)*NZ	4	12*(nx+ny)*NZ
After Communication Reduction	4	12*(nx+ny+4)*NZ	No Communication	

In addition to saving time with smaller total message size used in GPU to GPU data communication, this new communication reduction method also reduces the overhead time caused both by data transfers between GPU and CPU, and by MPI communication initialization. To illustrate the improvements in terms of timing, consider the following case. Suppose the overhead of MPI initialization takes t1, and data copying between CPU and GPU takes t2 amount of time, our communication reduction saves an additional $4*t1+8*t2$ amount of time. This is because four of eight MPI communications are replaced, and as a result four instances of data copying from GPU to CPU and from CPU back to GPU are removed. Moreover, since there is no need for data communication for stress computation, more time is available for overlapping communication with computation. With the communication reduction method, we can overlap communications with both velocity and stress computation. The following section will discuss this aspect in more detail.

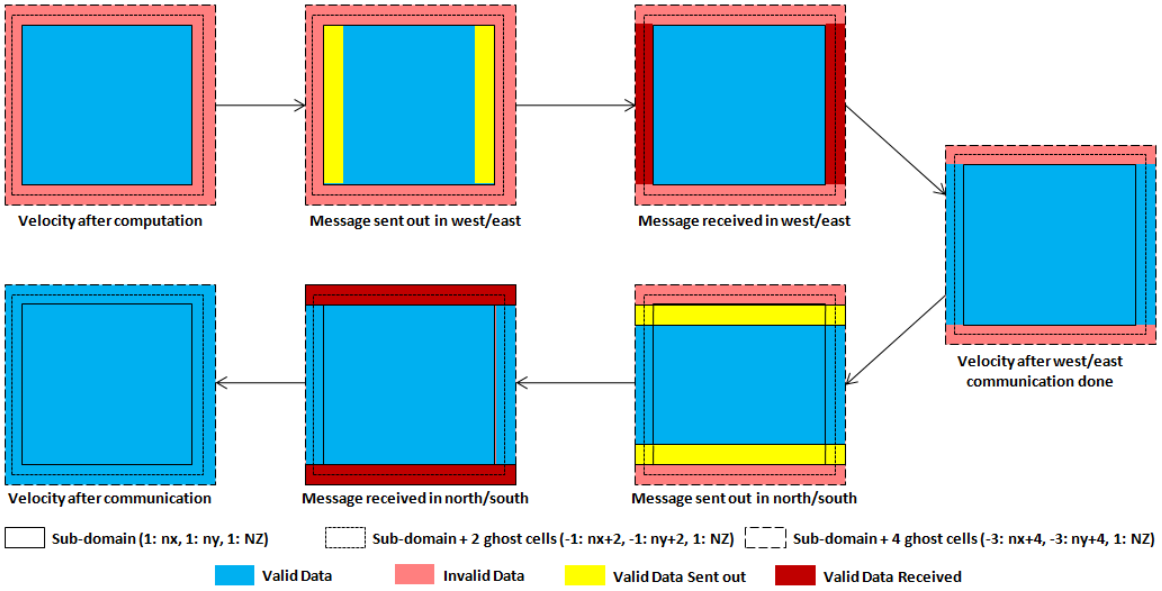

Fig.3. In-order communication: fills four layer ghost cells required by communication reduction and keeps the MPI communication frequency at four times per iteration.

3.3 In-order MPI Communication

In 13-point stencil computation, information needed for computation does not come from diagonal mesh points. Hence we can neglect the information about the corners in the two-layer ghost cell regions. In AWP-ODC-CPU code, there is no data shared between messages of a sub-domain. Hence any MPI scheduling can be adopted, including simultaneous data swap in all directions. However, the communication reduction method we introduce in the GPU code requires two extra ghost cells. For the calculation of the first two layers of ghost cells, we need all four corners' information. If we used the communication schedule of AWP-ODC-CPU, then we would need 4 additional MPI data swaps for corners, which would bring new overhead. To solve this problem, we implement an in-order communication method so not to increase the number of MPI messages. Figure 3 describes this method: we first send data in yellow regions (1: 4, 1: ny, 1: NZ) and (nx-3: nx, 1: ny, 1: NZ) to fill the data in red ghost cell regions (-3: 0, 1: ny, 1: NZ) and (nx+1: nx+4, 1: ny, 1: NZ) in west/east neighboring sub-domains. After the communication is done in the west/east directions, the data in (-3: nx+4, 1: ny, 1: NZ) becomes valid. We then send the yellow regions on north/south boundaries (-3: nx+4, 1: 4, 1: NZ) and (-3: nx+4, ny-3: ny, 1: NZ) to fill the ghost cell region (-3: nx+4, -3: 0, 1: NZ) and (-3: nx+4, ny+1: ny + 4, 1: NZ) in north/south neighboring sub-domains. After the in-order communication, the velocity data in the whole sub-domain (-3: nx+4, -3: ny+4, 1: NZ) is valid for the stress computation.

3.4 Enhanced Computation/Communication Overlapping

The communication reduction helps reduce the frequency of data communication, and the in-order communication guarantees the accuracy of the data for computation. Another key point is to design an effective computing/communication schedule to hide the communication latency. Since we only need to exchange velocity data in the four-layer ghost cells region, our idea is to use the total time spent on the computation of velocity and stress to overlap the entire communication latency, i.e. data copying between CPU and GPU, and MPI communications between CPUs. To present this approach, we define some regions and symbols in Figure 4 and Table 2.

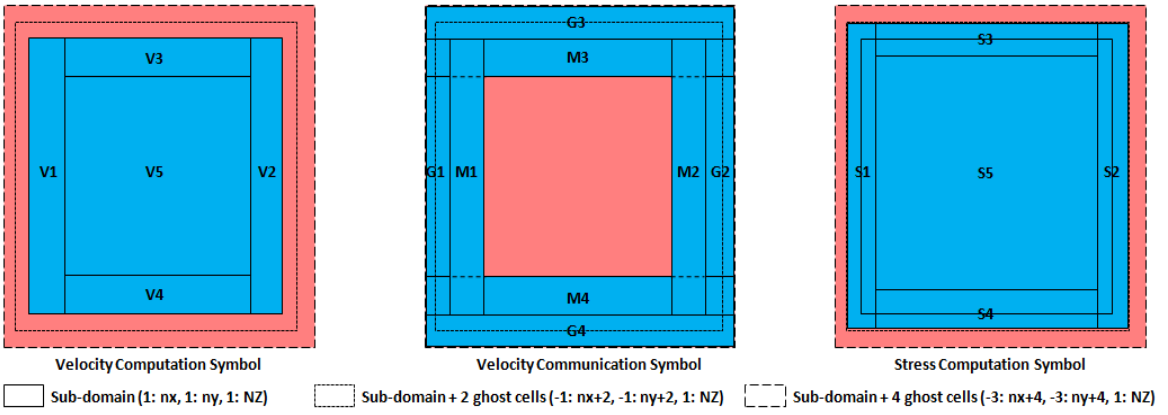

Fig.4. Definitions of regions and symbols for the enhanced overlapping algorithm

Figure 5 illustrates the flow of our effective computation and communication overlapping algorithm. First, the GPU starts to compute the velocity on the X-axis boundary, V1 and V2, in serial, and copies the resulting data to the CPU as soon as it is available using CUDA asynchronous memory copy. The CPU sends the velocity data M1 and M2 using asynchronous "MPI_ISend" when copying of V1 and V2 to the CPU is done. While the velocity data V1 and V2 are being copied to the CPU, the GPU computes the velocity on the Y-axis boundary, V3 and V4, simultaneously. After M1 & M2 MPI communications are done, the code schedules similar asynchronous data copying to send V3 and V4 to the CPU and send the M3 & M4 MPI messages afterwards.After all velocity computation is completed for the boundary, the GPU focuses on the computation of V5. Thus the velocity data communication time for the boundaries, including data copying from GPU to CPU and MPI communications, can be partially overlapped with the velocity computation.

Table 2. Description of the symbols defined in Figure 4.

Symbols	Description	Region
V1/V2	Velocity west/east 4 layers in non-ghost cell region	(1: 4, 1: ny, 1: NZ) / (nx-3: nx, 1: ny, 1: NZ)
V3/V4	Velocity north/south 4 layers in non-ghost cell region	(5: nx-4, 1: 4, 1: NZ)/ (5: nx-4, ny-3: ny, 1: NZ)
V5	The rest of the velocity computation in non-ghost cell region	(5: nx-4, 5: ny-4, 1: NZ)
M1/M2	Velocity west/east 4 layers message sent in west/east	(1: 4, 1: ny, 1: NZ) / (nx-3: nx, 1: ny, 1: NZ)
M3/M4	Velocity north/south 4 layers message sent in north/south	(-3: nx+4, 1: 4, 1: NZ) / (-3: nx+4, ny-3: ny, 1: NZ)
G1/G2	Velocity west/east 4 layers message received in west/east	(-3: 0, 1: ny, 1: NZ) / (nx+1: nx+4, 1: ny, 1: NZ)
G3/G4	Velocity north/south 4 layers message received in north/south	(-3: nx+4, -3: 0, 1: NZ) / (-3: nx+4, ny+1: ny+4, 1: NZ)
S1/S2	Stress west/east 4 layers including 2 ghost cell region	(-1: 2, -1: ny+2, 1: NZ)/ (nx-1: nx+2, -1: ny+2, 1: NZ)
S3/S4	Stress north/south 4 layers including 2 ghost cell region	(3: nx-2, -1: 2, 1: NZ) / (3: nx-2, ny-1: ny+2, 1: NZ)
S5	The rest of the stress computation in non-ghost cell region	(3: nx-2, 3: ny-2, 1: NZ)

As discussed earlier, the stress computation region is extended to (-1: nxt+2, -1: nyt+2, 1: NZ) from (1: nxt, 1: nyt, 1: NZ) as part of the communication reduction process. With this approach we do not need any communication for the stress components. In order to increase the overlapping time between communication and computation, we divide the stress region into five parts (S1 to S5) and reorder the computation schedule

from inside to outside. For the computation of inside stress region S5 (3: nx-2, 3: ny-2, 1: NZ), all the required information is the velocity for region (1: nx, 1: ny, 1: NZ). This information is available after the velocity computation is completed. Our strategy is to compute the inside region S5 first and then the boundaries S1-S4, so that the time spent on MPI communications for the velocity M1-M4 can be overlapped with the computation of S5. As soon as M1, M2, M3 or M4 of neighboring sub-domains is received, G1, G2, G3 or G4 will be available for initiating asynchronous data copying from CPU to GPU. Thus the time spent on data copying from CPU to GPU can also be overlapped with the stress computation. Moreover, some "multi-layer" overlapping may take place during the iterations. For example, S5 stress computation inside GPU, G2 data copying from CPU to GPU, and M3 MPI communication between CPUs may be running simultaneously. After the ghost cells data G1-G4 is copied from CPU to GPU, the GPU computes the stress for the rest of the sub-domain S1-S4 in order. In the end, the updated stress (-1: nxt+2, -1: nyt+2, 1: NZ) is ready for the next iteration.

AWP-ODC-GPU Main Loop:

Do T= timestep 0 **to** timestep N:

Pre-Post MPI_IRecv waiting for V1, V2, V3 and V4 of (vx, vy, vz).
Compute V1 for (vx, vy, vz) in GPU
Compute V2 for (vx, vy, vz) in GPU and initiate the transfer of V1 from GPU to CPU
Compute V3for (vx, vy, vz) in GPU and initiate the transfer of V2 from GPU to CPU
Compute V4 for (vx, vy, vz) in GPU
Compute V5 for (vx, vy, vz) in GPU
Compute S5 for (xx, yy, zz, xy, yz, xz) in GPU
Wait forV1/V2 data transfer done andtheninitiate MPI_ISend for M1/M2
Wait for M1, received MPI message and initiate the transfer of G1 from CPU to GPU
Wait forG1 data transfer done and then initiate the transfer of V3 from GPU to CPU
Wait forV3 data transfer done and then initiate MPI_ISend for M3
Wait for M2, received MPI message and initiate the transfer of G2 from CPU to GPU
Wait forG2 data transfer done and then initiate the transfer of V4 from GPU to CPU
Wait forV4 data transfer done and then initiate MPI_ISend for M4
Wait for M3/M4, received MPI message and initiate the transfer of G3/G4 from CPU to GPU
Compute the rest of stress computation S1-S4 for (xx, yy, zz, xy, yz, xz) in GPU

End Do

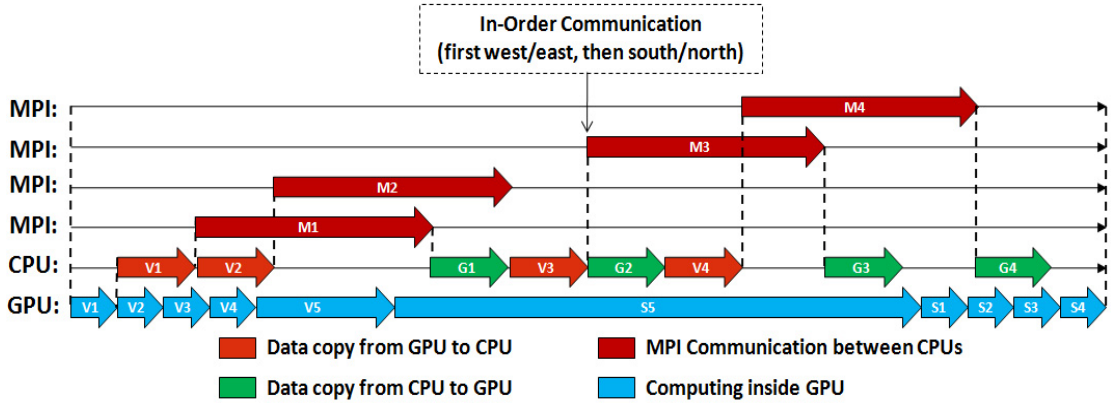

Fig.5. Effective computation and communication overlapping algorithm for AWP-ODC-GPU implementation.

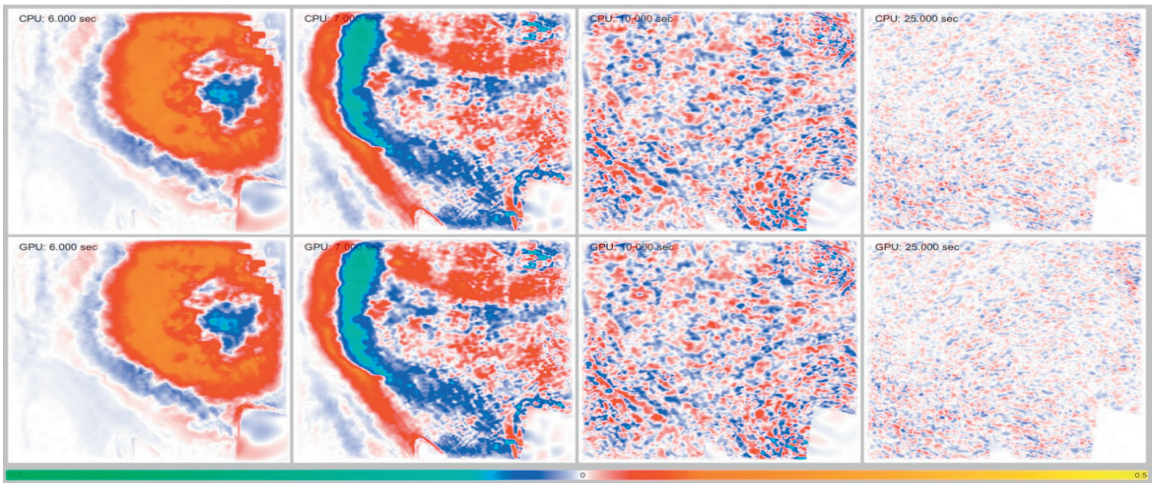

Fig.6. Comparison of visualizations of surface velocity in the X direction in units of m/s. Four snapshots are taken at seconds 6, 7, 10 and 25 (after 6,000, 7,000, 10,000, 25,000 timesteps respectively). The verification of AWP-ODC-CPU code has been done by geo-scientists for years. The south-east corner of the mesh includes an area with rocks. This uneven distribution of materials in the simulated area results in differences in the wave propagation compared to other parts of the map.

4. Experiments and Results

To verify the correctness of the multi-GPU version of the AWP-ODC code, we ran a large scale wave propagation simulation of the Mw5.4 Chino Hills, CA earthquake. This up-to-2.5Hz simulation has a mesh with dimension 1024 cubic. The total number of timesteps is 75,000, where each timestep simulates a period of 1millisecond. There are a total of 980 earthquake source points that are active for the duration of 2,500 timesteps (2.5 seconds). AWP-ODC-CPU code was run on the National Center for Atmospheric Research's (NCAR) Yellowstone CPU-based homogeneous supercomputer [23] using 512 CPU cores, while AWP-ODC-GPU code was run for the same simulation on the XSEDE Keeneland Full Scale System (KFS) using 128 GPUs. Figure 6 presents a heat map of the magnitude of the velocity in X direction to compare the two codes. The two identical simulation images show the accuracy of this new AWP-ODC-GPU code.

AWP-ODC-GPU was benchmarked for a weak scaling study on NCCS Titan Phase 5 system and XSEDE Keeneland Initial Delivery System (KIDS). Figure 7 shows the performance in GFlops achieved on the two heterogeneous supercomputers. The CUDA compiler is version 4.1 and the MPI compiler is Ohio State University's mvapich2 version 1.8, but the limic library (an I/O library used by mvapich2 compiler) support is disabled during the benchmark test. Only one NVIDIA X2090 GPU per node is available on Titan Phase 5 system, whereas 3 NVIDIA M2090 GPUs per node can be used on KIDS system. The full Titan Phase 5 system contains 960 GPUs whereas the full KIDS system contains 320 GPUs. In this benchmark, each GPU carries out stencil calculations for a mesh with size of 224x224x1024. The total number of points in the mesh becomes $224*224*1024*N*G$, where N represents the number of nodes and G represents the number of GPUs per node. The results are indistinguishable from ideal linear weak scaling since the performance in GFlops in single precision shows a linear increase with increased number of GPUs. On the KIDS system, using three GPUs per node achieves triple the GFlops compared to the case when we use one GPU per node. On the NCCS Titan Phase 5 system, the peak performance achieved is 101.4 TFlops for the mesh size 7616*6272*1024 (~ 49 billion mesh points) using 952 GPUs, while the peak computing performance achieved on KIDS system is 30.5 TFlops for the mesh size 4032*3360*1024 (~ 14 billion mesh points) using 270 GPUs. Linear weak scaling means our communication model hides communication latency successfully and provides a good solution to the bottleneck caused by the data transfer between CPU and GPU.

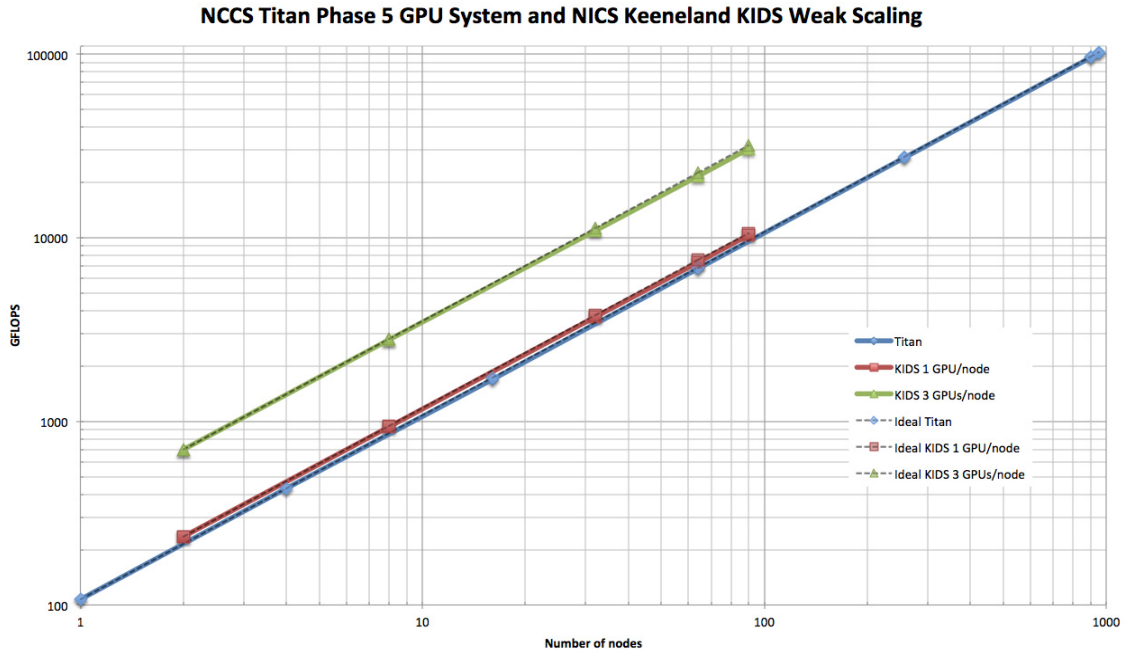

Fig.7. Weak scaling study on NCCS Titan Phase 5 system and XSEDE Keeneland KIDS in terms of GFlops achieved with respect to the number of nodes requested. Linear weak scaling was observed.

5. Conclusion and Future Work

Recent destructive earthquakes in China, Haiti, Chile, New Zealand, and Japan, highlight the national and international need for improved seismic hazard evaluation. Energy efficient high performance earthquake codes are urgently needed for this purpose. Toward this goal, we have developed a multi-GPU implementation of a highly scalable earthquake simulation code for heterogeneous supercomputers. This code was restructured to enable maximized throughput and efficiency for GPU systems. To avoid degrading the computational performance achieved by our previous single GPU optimization work, a notable communication model has been developed to hide the communication latency, especially to overlap the high latency caused by data communication between CPU and GPU. With this successful optimization approach, we demonstrated excellent weak scaling on both the NCCS Titan 5 system and the NICS Keeneland KIDS system, with sustained 100-TFlops using 952 Fermi GPUs for a 77-fold speedup.

Further enhancement of AWP-ODC-GPU is under way for efficient use of hybrid multi-core architectures such as Kepler-based NCCS Titan and NCSA Blue Waters. The long term goal is to use this code to calculate an improved probabilistic seismic hazard forecast for the California region, a collaborative effort coordinated by Southern California Earthquake Center.

6. Acknowledgements

This work used the Extreme Science and Engineering Discovery Environment (XSEDE) supported by National Science Foundation grant number OCI-1053575, and the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) supported by the Office of Science of the Department of Energy under contract DE-AC05-00OR22725. Simulations and benchmarks were performed on Titan Phase 5 system at ORNL and Georgia Tech Keeneland at NICS. The research received funding support through the NSF XSEDE Extended Collaborative Support Service (ECSS) program, University of California at San Diego Graduate

Program, National Center for Supercomputing Applications (NCSA) PRAC Direct Funding, and Southern California Earthquake Center's core program of 2012. The GPU implementation and testing were carried out on High Performance GeoComputing Lab's Fermi GPU cards donated by Nvidia. The authors acknowledge following individuals for their contributions to this work: Kim Olsen of San Diego State University for science advice, Carl Ponder of Nvidia for technical support, AmitChouraisa of SDSC for visualizations, Philip Maechling and Patrick Small of Southern California Earthquake Center for velocity model generation.

References

[1] K. B. Olsen, "Simulation of Three-Dimension Wave Propagation in the Salt Lake Basin". Doctoral Dissertation, University of Utah, 1994.

[2] Y. Cui, K. B. Olsen, T.H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. Panda, A. Chourasia, J. Levesque, S. M. Day and P. Maechling."Scalable Earthquake Simulation on Petascale Supercomputers", In Proceedings of the 2010 ACM/IEEE conference on Supercomputing, SC'10, pp.1-20, Nov. 2010.

[3] http://scec.usc.edu/research/cme/groups/cybershake

[4] J.C. Thibault and I. Senocak, "CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incomressible flows", in Proceedings of the 47th AIAA Aerospace Sciences Meeting, no. AIAA 2009-958, Jan. 2009.

[5] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori and M. Taiji. "42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence", in SC'2009: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. New York, NY, USA, ACM2009 pp.1-12.

[6] T.Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama and S. Matsuoka. "An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code", in SC'2010: Proceeding of the Conference on High Performance Computing Networking, Storage and Analysis. New Orleans, LA, USA, ACM2010, pp.1-11.

[7] http://www.top500.org/lists

[8] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukuda, S. Matsuoka. "Peta-scale Phase-field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer", in SC'2011: Proceeding of the Conference on High Performance Computing Networking, Storage and Analysis. Seattle, WA, USA, ACM 2011. pp.1-11.

[9] P. Micikevicius. "3D Finite-difference Computation on GPUs Using CUDA". In GPGPU-2: Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units, Washington, DC, USA, 2009, pp. 79-84.

[10] R. Abdelkhalek, H. Calandra, O. Coulaud, J. Roman, G. Latu. "Fast Seismic Modeling and Reverse Time Migration on a GPU Cluster". International Conference on High Performance Computing and Simulation, 2009, HPCS'09, Leipzig, Germany, pp.36-43. Aug. 07, 2009.

[11] D. Komatitsch, D. Michea, G. Erlebacher. "Porting a high-order Finite-element Earthquake Modeling Application to NVIDIA Graphic Cards Using CUDA". J. Parallel Distributed Comput. 69 (5) (2009) pp.451-460.

[12] D. Komatitsch, G. Erlebacher, D.Goddeke, D. Michea. "High-order Finite-element Sesimic Wave Propagation Modeling with MPI on a Large GPU Cluster". J. Comput. Phys. Vol. 229, Issue 20, Oct. 2010, pp.7692-7714.

[13] D. Komatitsch, D. Goddeke, G. Erlebacher, D. Michea. "Modeling the Propagation of Elastic Waves Using Spectral Elements on a Cluster of 192 GPUs". Computer Science – Research and Development. Vol. 25, No. 1-2. pp. 75-82.

[14] D. Michea, D. Komatitisch. "Accelerating a three-dimensional finite-difference Wave Propogation Code Using GPU Grapics Cards". Geophys. J. Int. 182(1) (2010), pp. 389-402

[15] S. Song, T. Dong, Y. Zhou, D. Yuan, and Z. Lu. "Sesmic Wave Propagation Simulation Using Support Operator Method on Multi-GPU System". In Technical Report, University of Minnesota, 2010.

[16] T. Okamoto,H. Takenaka, T. Nakamura, and T. Aoki."Accelerating large-scale simulation of seismic wave propagation bymulti-GPUs and three-dimensional domain decomposition", Earth Planets and Space, 62, pp.939-942, 2010

[17] T. Okamoto, H. Takenaka, T. Nakamura, and T. Aoki."Accelerating Simulation of Seismic Wave Propagation by Multi-GPUs". AGU 2010 Meeting, San Francisco, California, Dec. 13-17, 2010.

[18] T. Okamoto, H. Takenaka, T. Hara, T. Nakamura, and T. Aoki."Rupture Process And Waveform Modeling of The 2011 Tohoku-Oki, Magnitude-9 Earthquake". AGU 2011, San Francisco, California, Dec. 5-9, 2011.

[19] J. Zhou, D. Unat, D. J. Choi, C. C. Guest and Y. Cui. "Hands-on Performance Tuning of 3D Finite Difference Earthquake Simulation on GPU Fermi Chipset", Procedia Computer Science, vol. 9, pp. 976-985, 2012.

[20] http://www.olcf.ornl.gov/titan/titan-overview/

[21] http://www.cray.com/Products/XK6/XK6.aspx, Cray Company.

[22] NICS Keeneland Supercomputer: http://keeneland.gatech.edu/

[23] NCAR Yellowstone Supercomputer: https://www2.cisl.ucar.edu/resources/yellowstone