UNIVERSITY OF CALIFORNIA, SAN DIEGO

Scalable Parallel Programming for High Performance Seismic Simulation on Petascale Heterogeneous Supercomputers

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering
(Computer Engineering)

by

Jun Zhou

Committee in charge:

Professor Clark C. Guest, Chair
Professor Yifeng Cui
Professor Chung-Kuan Cheng
Professor William S. Hodgkiss
Professor Vitaliy Lomakin
Professor Jean-Bernard Minster

2014

This dissertation of Jun Zhou is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2014

DEDICATION

To my dear father, mother and wife.

EPIGRAPH

Believe you can and you're halfway there.

- *Theodore Roosevelt*

TABLES OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

In the first place, I would like to thank my advisor Professor Clark C. Guest, for his extreme patience, tireless encouragement and generous support through my Ph.D. study. His knowledge, wisdom, and principles inspired me from the very beginning of my research. He provided me with freedom to choose research topics and work on my own pace. He helped me to grow up from a fresh graduate student to a professional researcher. His invaluable guidance and dedication made this dissertation a reality. It has been a great pleasure working with him as his Ph.D. student.

I would also like to express my gratitude to my co-advisor Professor Yifeng Cui. He is a pioneer in high performance computing area and his invaluable insights always kept my research on the right track. I really appreciate his talent, help and generous expertise sharing. He provided me tremendous resource and gave me 100% trust. His advice has been always prompt and effective. He was and remains my best role model for a scientist and mentor. This dissertation would not have been possible to finish without his fully support. Special thanks to Sanhong Liu, the wife of Professor Yifeng Cui, for inviting me to enjoy her awesome homemade Chinese food and sharing her and Professor Yifeng Cui's funny stories. Many thanks for all the memories.

I am very thankful to other committee members: Professor Chung-Kuan Cheng, Professor William S. Hodgkiss, Professor Vitaliy Lomakin, and Professor Jean-Bernard Minster. I am really lucky to have these world-class professors and researchers in my committee. I would also like to thanks my undergraduate and master advisors Professor Guangda Su and Professor Yaolin Tan at Department of Electrical Engineering, Tsinghua University in China, who helped me build up a solid fundamental knowledge and provided me with motivation to pursue Ph.D.

I am truly indebted to my lab fellows; Kwangyoon Lee, Shiyu Song and Efecan Poyraz, who provided me many helpful insights and constructive perspectives in both technical and non-technical parts. I am especially thankful to Efecan Poyraz for helping me debug the code/program and reviewing my papers/dissertation in details. It is my pleasure to thank Dr. Dong Ju Choi, Amit Chourasia, Dr. Joey Reed, Dr. Hieu Nyugen and Dr. Didem Unat for their collaboration. I gratefully acknowledge many friends during my study at UCSD; Shaohe Wang, Minghai Qin, Yangbin Gao, Weixin Li, Menglai Han, Yichao Huang, Lele Wang, Lelin Zhang,

Xiang Hu, Yulei Zhang, Yufei Shi, Weiyi Lu, Yun Liang, Kan Yu, Jean Fan, Siming Wang, Minghui Zhu. I would never have survived without these incredible friends.

Finally, I would like to thank my parents for their unconditional support. Mom and Dad, thank you for bringing me up and allowing me to study and live in a different country far from China to pursue my dream. I especially thank to my best friend, soul-mate and loving wife, Serena Fan Wang. She always has faith in me and has been a great supporter during my good and bad times. I owe sincere thankfulness to her for invaluable love and sticking by my side for last three years. There are no words to convoy how important you are in my life. I also want to thank my parents-law for letting their daughter marry me.

Some sections of this dissertation are based on published papers which I have co-authored with others:

Section 3.2, Section 4.2 and Section 5.1, are based on the material as it partly appears in proceedings of the 2010 ACM/IEEE conference on Supercomputing with title "Scalable Earthquake Simulation on Petascale Supercomputers" by Yifeng Cui, Kim B. Olsen, Thomas H. Jordan, Kwangyoon Lee, Jun Zhou, Patrick Small, Daniel Roten, Geoffrey Ely, Dhabaleswar K. Panda, Amit Chourasia, John Levesque, Steven M. Day and Philip Maechling. I was one of the primary investigators and author of this paper.

Section 3.1 and 3.3, in part, are a reprint of the material as it appears in the 2010 third international joint conference on Computational Science and Optimization with title "Workflow-Based High Performance Data Transfer and Ingestion to Support Petascale Simulations" by Jun Zhou, Yifeng Cui, Sashka Davis, Clark C. Guest, Philip Maechling. I was one of the primary investigators and author of this paper.

Section 4.1 and 4.3-4.5, in part, are a reprint of the material as it appears in the 2012 international conference on Computational Science with title "Hands-on Performance Tuning of 3D Finite Difference Earthquake Simulation on GPU Fermi Chipset" by Jun Zhou, Didem Unat, Dong Ju Choi, Clark C. Guest and Yifeng Cui. I was one of the primary investigators and author of this paper.

Section 5.2 and 5.3, in part, are a reprint of the material as it appears in the 2013 international conference on Computational Science with title "Multi-GPU Implementation of a 3D Finite Difference Time Domain Earthquake Code on Heterogeneous Supercomputers" by Jun

Zhou, Yifeng Cui, Efecan Poyraz, Dong Ju Choi, and Clark C. Guest. I was one of the primary investigators and author of this paper.

Section 5.3 and Chapter 6 in part, are based on the material as it partly appears in proceedings of the 2013 ACM/IEEE conference on Supercomputing with title "Physics-based Seismic Hazard Analysis on PetaScale Heterogeneous Supercomputers" by Yifeng Cui, Efecan Poyraz, Kim B. Olsen, Jun Zhou, Kyle Withers, Scott Callaghan, Jeff Larkin, Clark C. Guest, Dong Ju Choi, Amit Chourasia, Steven M. Day, Philip Maechling, and Thomas H. Jordan. I was one of the primary investigators and author of this paper.

VITA

2005	Bachelor of Engineering, Tsinghua University, China
2008	Master of Engineering, Tsinghua University, China
2014	Doctor of Philosophy, University of California San Diego, United States

PUBLICATIONS

J. Zhou, Y. Cui, S. Davis, C. C. Guest, P. Maechling, "Workflow-based high performance data transfer and ingestion to petascale simulations on TeraGrid", IEEE Computational Sciences and Optimization (CSO10), vol. 1, pp. 343-347, 2010.

Y. Cui, K.B. Olsen, T.H. Jordan, K. Lee, J. Zhou, P. Small, G. Ely, D. Roten, DK 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'2010, ACM Gordon Bell Finalist, New Orleans, Nov, 13-19, 2010.

D. Unat, J. Zhou, Y. Cui, X. Cai and S. Baden, "Accelerating an Earthquake Simulation with a C-to-CUDA Translator", Journal of Computing in Science and Engineering, vol. 14, No. 3, 48-58, May/June, CiSESI-2011-09-0094, May, 2012.

J. Zhou, D. Unat, D. Choi, C. C. Guest, Y. Cui, "Hands-on Performance Tuning of 3D Finite Difference Earthquake Simulation on GPU Fermi Chipset", Proceedings of International Conference on Computational Science, Vol. 9, 976-985, Elesvier, ICCS'2012, Omaha, Nebraska, June 4-6, 2012.

J. Zhou, Y. Cui, E. Poyraz, D. Choi, C. C. Guest, "Multi-GPU Implementation of a 3D Finite Difference Time Domain Earthquake Code on Heterogeneous Supercomputers", Proceedings of International Conference on Computational Science, Vol. 18, 1255-1264, Elsevier, ICCS'2013, Barcelona, June 5-7, 2013.

Y. Cui, E. Poyraz, K.B. Olsen, J. Zhou, K. Withers, S. Callaghan, J. Larkin, C.C. Guest, D. Choi, A. Chourasia, Z. Shi, S.M. Day, P. Maechling and T.H. Jordan, "Physics-based Seismic Hazard Analysis on Petascale Heterogeneous Supercomputers", In Proceedings of the 2013 ACM/IEEE conference on Supercomputing, SC'2013, Denver, Nov 17-22, 2013.

Y. Cui, E. Poyraz, J. Zhou, S. Callaghan, P. Maechling, P. Chen and T. H. Jordan, "Accelerating CyberShake Calculations on XE6/XK7 Platforms of Blue Waters", Extreme Scaling Workshop 2013, August 15-16, Boulder, 2013.

ABSTRACT OF THE DISSERTATION

Scalable Parallel Programming for High Performance Seismic Simulation on Petascale Heterogeneous Supercomputers

by

Jun Zhou

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2014

Professor Clark C. Guest, Chair

The 1994 Northridge earthquake in Los Angeles, California, killed 57 people, injured over 8,700 and caused an estimated $20 billion in damage. Petascale simulations are needed in California and elsewhere to provide society with a better understanding of the rupture and wave dynamics of the largest earthquakes at shaking frequencies required to engineer safe structures. As the heterogeneous supercomputing infrastructures are becoming more common, numerical developments in earthquake system research are particularly challenged by the dependence on the accelerator elements to enable “the Big One” simulations with higher frequency and finer resolution. Reducing time to solution and power consumption are two primary focus area today for the enabling technology of fault rupture dynamics and seismic wave propagation in realistic 3D models of the crust’s heterogeneous structure.

This dissertation presents scalable parallel programming techniques for high performance seismic simulation running on petascale heterogeneous supercomputers. A real world earthquake simulation code, AWP-ODC, one of the most advanced earthquake codes to date, was chosen as the base code in this research, and the testbed is based on Titan at Oak Ridge National Laboraratory, the world's largest hetergeneous supercomputer. The research work is primarily related to architecture study, computation performance tuning and software system scalability. An earthquake simulation workflow has also been developed to support the efficient production sets of simulations. The highlights of the technical development are an aggressive performance optimization focusing on data locality and a notable data communication model that hides the data communication latency. This development results in the optimal computation efficiency and throughput for the 13-point stencil code on heterogeneous systems, which can be extended to general high-order stencil codes. Started from scratch, the hybrid CPU/GPU version of AWP-ODC code is ready now for real world petascale earthquake simulations. This GPU-based code has demonstrated excellent weak scaling up to the full Titan scale and achieved 2.3 PetaFLOPs sustained computation performance in single precision. The production simulation demonstrated the first 0-10Hz deterministic rough fault simulation. Using the accelerated AWP-ODC, Southern California Earthquake Center (SCEC) has recently created the physics-based probablistic seismic hazard analysis model of the Los Angeles region, CyberShake 14.2, as of the time of the dissertation writing. The tensor-valued wavefield code based on this GPU research has dramatically reduced time-to-solution, making a statewide hazard model a goal reachable with existing heterogeneous supercomputers.

Chapter 1

Introduction

The No.1 supercomputer Tianhe-2, based on the Top 500 supercomputer list released in November 2013, has achieved 54.9 PFLOPs theoretical peak performance. The computing architectures today become increasingly more heterogeneous, over 30 supercomputers from the November list, in fact, exceed PetaFLOPs computation capability [1]. The computing node of these giant machines contains both general-purpose processors and specialized massively parallel processors - Graphics Processing Units (GPU). In recent years, a notable portion of large-scale scientific applications have taken advantage of the availability of the heterogeneous computing architectures. The 2011 Gordon Bell Prize winner, for example, is a phase-field simulation on TSUBAME 2.0 supercomputer [2], and four of six 2013 Gordon Bell Prize Finalists in 2013 used Titan Supercomputer [3]. The goal of this dissertation is to take physics-based seismic simulation to a new level to enable petascale computation on the hybrid heterogeneous systems.

Earthquake System is one of the most important research challenges to human beings, which is to provide society with a better understanding of earthquake causes and effects, with the goal of reducing the potential of loss of lives or properties. A variety of numerical methods have been utilized for modeling earthquake simulation, such as Finite Difference, Finite Element, and Spectral Element. In recent years, many of those earthquake simulations become more and more data intensive, and demand not only computing capability and accuracy, but also fast and effective computation efficiency. Probabilistic seismic hazard analysis (PSHA) has been effective in helping decision-makers reduce seismic risk and increase community resilience. However, the earthquake threat is highly time-dependent. To understand risk and improve resilience, we need to quantify earthquake hazards in physics-based models that can be coupled to engineering models of the built environment. The enabling technology of earthquake system science and physics-based PSHA is the numerical simulation of fault rupture dynamics and seismic wave propagation in realistic 3D models of the crust's heterogeneous structure.

AWP-ODC, which stands for Anelastic Wave Propagation by Olsen, Day and Cui, is a 3D Finite Difference Time Domain (FDTD) earthquake simulation code [4] used by Southern

California Earthquake Center (SCEC) for large-scale earthquake simulations. In 2010, the AWP-ODC code achieved a record earthquake simulation named "M8": a full dynamic simulation of a magnitude8 earthquake on the southern San Andreas fault with a time sample rate up to 2Hz. The M8simulation produced 360 seconds of wave propagation in an 810km by 405km Southern California area, and calculated 436 billion mesh points on a uniform grid with a resolution of 40m. The M8 achieved 220 TFLOPS for 24 hours on Cray XT5 Jaguar machine at ORNL using 223,074CPU cores, an ACM Gordon Bell finalist [5]. This code has been adapted for the reciprocity-based CyberShake calculation, a project aiming at producing a California state-wide physic-based hazard map [6]. Accelerating this wave propagation engine is with a goal of dramatically reducing hundreds of millions of allocation hours otherwise needed to generate such a map at frequency as large as 1-Hz.

Despite the improvements in heterogeneous computing driven by the increased attention from the HPC community, there is a significant burden for porting FDTD simulations to CPU-GPU heterogeneous supercomputers. FDTD applications are well known for their low sustained FLOPS due to frequent data communication required between the CPU and GPU. For this reason, considerable room remains to tune the application performance on the latest hybrid architectures. This dissertation presents a multi-GPU implementation of AWP-ODC, and describes details of the tuning techniques used for efficient CPU-GPU heterogeneous supercomputers. Re-structured from the Fortran-based AWP-ODC CPU code, the GPU-based AWP-ODC is written in C before CUDA and MPI are implemented for a hardware-oriented design for achieving high performance. To avoid the data communication latency due to the slow PCIe connection, our strategy is to extend the computing region on accelerators and decrease the number and size of data to be exchanged as much as possible, in order to hide the data communication time. The algorithms utilized are highly scalable because they are carefully arranged to hide latency of computation and communication, resulting in optimal speedup and maximized parallel efficiency. A communication model is implemented to reduce the intra-node frequency of data movement between CPU and GPU, which allows complete overlap of communication and computation. This model can be extended to general stencil computing on a structured grid. This GPU-based AWP-ODC code demonstrates tremendous performance improvement, which takes only 5.5 hours to finish a 440 billion mesh points scale earthquake simulation using 16K GPUs, the largest ever earthquake simulation performed. The new code is

carefully validated against both reference models and earthquake observations before being used for large-scale wave propagation simulations by SCEC.

The new AWP-ODC production code demonstrates perfect weak scaling on Titan, the world's largest GPU-based supercomputer. A sustained 2.3 PFLOPs performance in single precision, the code is the most scalable 3D seismic modeling code to date, particularly remarkable when considering the 3D computations of data-intensive memory-bound stencils. The new AWP-ODC also provides seismology scientists, for the first time, with ability to simulate ground motions with large fault ruptures to frequencies as high as 10 Hz in a physically realistic way. Most importantly, the new AWP-ODC capability of accelerating strain Green Tensor calculations promises a saving of more than 500 million computing core-hours required to generate the scheduled California state-wide CyberShake 1.0-Hz seismic hazard map.

Thesis Contributions

- Implementation of a highly scalable and efficient AWP-ODC earthquake simulation code for CPU-GPU based heterogeneous supercomputers. This new code is redesigned to allow optimal performance on the hybrid system. The enhanced computation and communication model effectively hide the data communication latency between CPU/GPU and between computing nodes. This development results in perfect weak scaling on full Titan scale, with a 2.3 PFLOPs sustained performance measured in single precision.
- Development of an earthquake simulation workflow running on the XSEDE, which supports petascale sets of simulations effectively using XSEDE resources, including high performance data transfer between supercomputers and data management for tens of thousands of Gigabyte input/output files.
- Demonstrated capability of a first 10-Hz deterministic earthquake simulation by the time of the dissertation writing. The new AWP-ODC code is extended to support the SCEC CyberShake project with dramatically reduction in time-to-solution and energy cost saving, recognized with my HPGeoC Lab colleagues together for the IDC HPC Innovation Excellence Award during ISC'2013 [7].
- The development of this GPU-based AWP-ODC code provides a solid basis prepared for upcoming many-core architectures and programming models. This seismic code, for example, can be restructured as coarse-grain for the 3D domain and fine-grain for a 2D

surface to take advantage of NVIDIA Kepler dynamic parallelism. The original AWP-ODC CPU code can be also tuned on the Intel MIC processor, using the high throughput, data locality and communication reduction schemes developed through this work.

Thesis Outline

- Chapter 2 provides some background and motivation for the dissertation. It describes the current heterogeneous supercomputer architectures and also parallel programming models for current systems. The chapter also presents some well-known scientific applications benefits from the heterogeneous architectures.
- Chapter 3 introduces the earthquake simulation workflow. Before introducing the workflow, we discuss why the workflow is necessary for large scale earthquake simulations and some technical background. Next, the chapter presents the workflow framework and provides details of its implementation.
- Chapter 4 presents the single GPU optimization of AWP-ODC. First, it describes our deep analysis of AWP-ODC numerical model, showing its characteristics as a 3D 13-point stencil computation with data-intensive memory-bound. Then the chapter provides the details of our hardware-based optimization and the code validatation.
- Chapter 5 presents the multi-GPU optimization of AWP-ODC. The chapter first describes the original communication model implementation on the CPU-based homogeneous supercomputer, then focuses on our enhanced communication model implementation for CPU-GPU based heterogenenous supercomputers, designed to reduce communication frequency between CPU/GPU and between computing nodes, and also fully hide the communication latency. Some code validation and benchmark experiments on OLCF Titan are presented.
- Chapter 6 describes the first 10-Hz deterministic earthquake simulation using our new AWP-ODC code running on the OLCF Titan machine. Then our CyberShake simulation support and validation is presented, along with some analysis of the computation time reduction and power saving for the future CyberShake project.
- Chapter 7 discusses future directions for high performance earthquake simulation using new architectures, and concludes the dissertation.

Chapter 2

Motivation and Background

This chapter describes the current heterogeneous supercomputers and typical stencil computation applications. Section 2.1 provides background information on supercomputers and current heterogeneous architectures utilized in the High Performance Computing (HPC) area, with an emphasis on the combination of Central Processing Units (CPUs) and Graphic Processing Units (GPUs). Section 2.2 introduces popular parallel programming models for supercomputers, including OpenMP (Open Multiprocessing), MPI (Message Passing Interface), CUDA (Compute Unified Device Architecture), OpenCL (Open Computing Language), and a developing parallel programming model OpenACC. Section 2.3 describes some world-class large scale stencil computation applications in recent years, and their sustained petascale performance results running on the Top 500 supercomputers. Section 2.4 presents a summary and conclusion.

2.1 Heterogeneous Supercomputers

"The number of transistors per integrated circuit will double every 18-24 months" [8], this famous Moore's Law has been correct in semi-conductor area for almost 30 years. Then in 2005, the maximum frequency of 3.8 GHz for a single CPU was reached and chip vendors could not achieve a higher number because of the "power wall" [9]: the power density is proportional to the cube of the frequency. Therefore, chip vendors chose to integrate two or more processing cores using lower frequency into a combined computing unit to provide higher performance. Traditional CPUs with more than one core are referred to as general-purpose multi-core processors. They can run any kind of applications including operating systems and database software. Currently most top 500 supercomputers are installed with this kind of processor [1], where each node contains one or two general-purpose multi-core processors. We call them "homogenous supercomputers", because each computing node has exactly the same multi-core architecture.

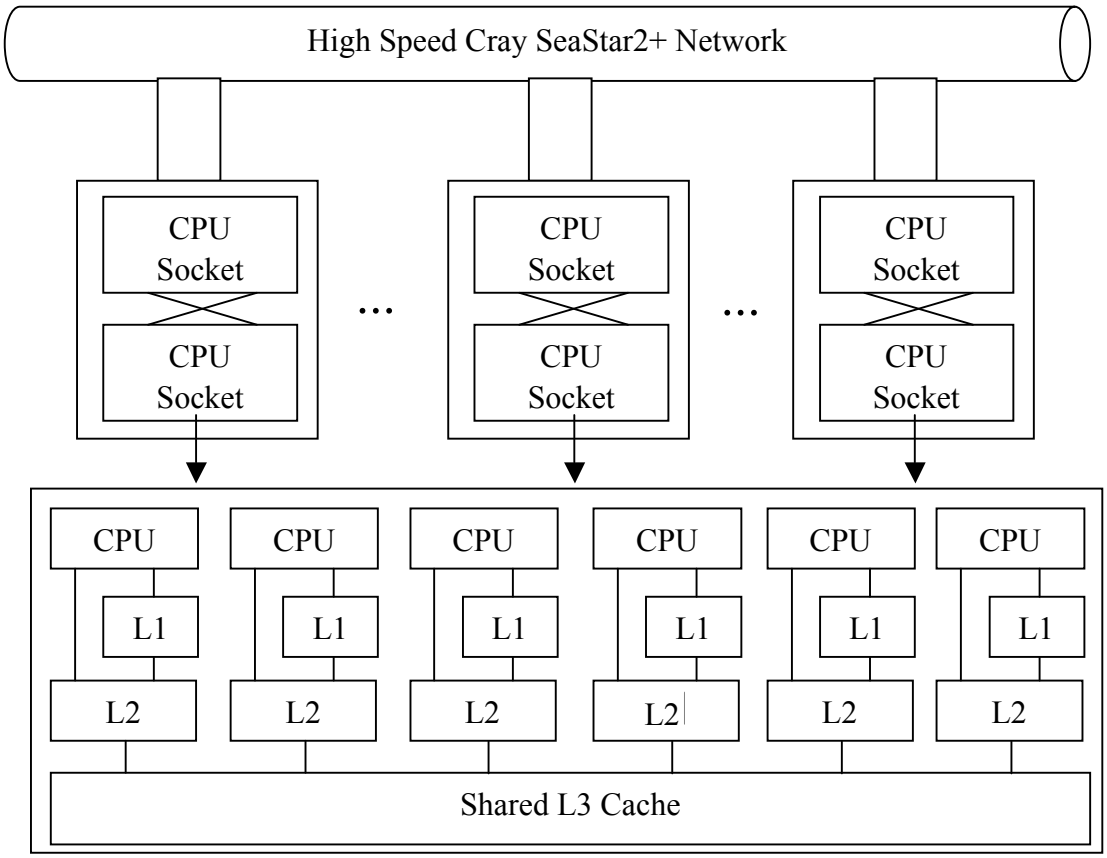


Figure 2.1: The homogenous supercomputer, "Kraken", with each computing nodes containing two general-purpose CPU sockets and six cores per socket [10].

Figure 2.1 shows one advanced NSF homogenous supercomputer, "Kraken", installed at the National Institute for Computational Sciences (NICS), currently ranked at 35 on the Top 500 list released in Nov. 2013 [1]. This supercomputer is a member of XSEDE (Extreme Science and Engineering Discovery Environment) to support large scale scientific computation and delivers more than 700 million CPU hours per year. The NICS Kraken is a Cray XT-5 system with 9,408 computing nodes and 1.17 PetaFlops peak computing capability. The interconnection is a Cray SeaStar2+ router. Each node of the Kraken contains two AMD Opteron processors (Istanbul) with 12 cores in total. Each AMD multi-core socket has a private 64KB L1 and 512KB L2 cache and shares a 6MB L3 cache with other CPU cores on the same socket. The clock frequency is 2.6 GHz and the total memory size shared by all six CPU cores is 8 GB. The shared memory control design is a NUMA (Non-Uniform Memory Access) architecture that maps different memory banks to different cores [10].

The heterogeneous supercomputer has a very similar architecture to the homogenous supercomputer. The primary difference is that each node of a heterogeneous supercomputer is a combination of a general-purpose CPU and accelerators, instead of a CPU only. Two heterogeneous architectures have been used to build supercomputers: Cell Broadband Engine Architecture, and a combination of a general-purpose CPU and one or more GPU [11].

The IBM Roadrunner supercomputer located at Los Alamo National Laboratory is based on CBEA with a peak performance of 1.7 PetaFlops. It was the first heterogeneous supercomputer and also the first Top 500 supercomputer that achieved a sustained 1.0 PetaFlops using the Linpack benchmark test [12]. Roadrunner has 6,912 AMD Opteron dual-core processors and 12,960 IBM PowerXCell 8i™ processors as computational accelerators [13]. The high speed interconnection network is Infiniband. Each node of Roadrunner is a hybrid design, containing two AMD Opteron processors plus one IBM PowerXCell 8i processor attached to each of them (See Figure 2.2).

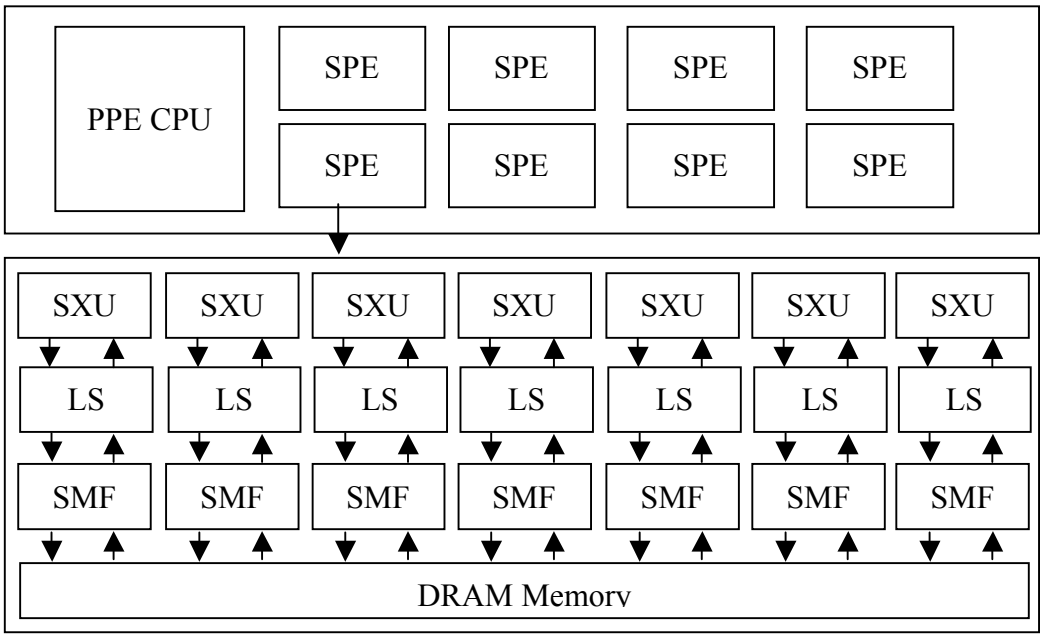


Figure 2.2: Schematic IBM PowerXCell 8i architecture (Cell Processor): consisting of one Power Processing Engine (PPE) CPU and eight Synergistic Processing Elements (SPE). SPE is a SIMD architecture (Single Instruction Multiple Data). Each SPE has eight processing units called SXU (Synergistic Execution Units). Each SXU has a private 256K local memory (LS) and a DMA memory engine SMF (Synergistic Memory Flow). One single IBM PowerXCell 8i chip has a theoretical 102.4 Gigaflops peak performance and 25.6 GB/sec memory bandwidth [14].

Currently, most heterogeneous supercomputers are not using CBEA, but rather the combination of general-purpose CPU and modern programmable GPU, which is very popular and widespread deployed for large-scale scientific computation. The latest Top 500 list released in Nov. 2013 shows two out of top 10 supercomputers are based on this architecture [1]. Before our introduction of these heterogeneous supercomputers, we briefly go over modern GPU design.

The adoption of Graphic Processing Units for high performance computing starts from 2007, when the NVIDIA Company released the CUDA language for GPU computing [15]. Before that, GPUs were designed for vertices-to-pixels transformation, and rendering images on the screen. Their main function was to support and accelerate image or graphic APIs (Application Programming Interfaces) from OpenGL or DirectX. But now, GPUs have already evolved into powerful multi-core systems with super high parallel computing capability that can handle large scale data parallel processing more efficiently than the CPU. In addition to their original multimedia application, GPUs have provided significant performance improvement in many computation intensive scientific or industrial applications such as astrophysics, climate research, earthquake simulation, bioinformatics, financial computing, oil industry and etc [16].

GPUs are not general purpose but focused on applications with high parallelism, and a CPU is also required to control the GPU for data management and non-parallel computing. On desktops or servers, the CPU and the GPU are separate chips connected by a PCI Express bus (Peripheral Component Interconnect Express). PCIe is very slow compared to the computations taking place, so data movement back and forth between CPU and GPU during computing will produce a serious performance penalty and should be avoided. On mobile devices, the CPU and the GPU are built on the same chip and share global memory, so data movement overhead can be avoided by memory mapping. For HPC on supercomputers, we concentrate on the GPU computing on server, thus we have to deal with the bottleneck caused by slow data movement, which will be discussed in a later section.

NVIDIA and AMD are the two largest vendors for the GPU computing, the NVIDIA Tesla is more popular as a GPU accelerator for large scale scientific computation. In the Top 500 list released in Nov. 2013, the No.12 Tianhe-1A machine in Tianjin, China and the No.21 Nebulae in Shenzhen, China have chosen NVIDIA Telsa M2050 and provide 4.70 PetaFlops and 2.98 PetaFlops respectively [1]. OLCF Titan is the largest CPU-GPU cluster in 2013/2014 ranked as No.2. Originally installed with the NVIDIA Tesla M2090, it has been upgraded by the latest

generation NVIDIA Kepler chipset and already achieved the theoretical peak performance with 27.1 PetaFlops, which can be fully in production use by researchers starting from the early of 2014 [17].

NVIDIA Tesla is the first dedicated "general purpose" GPU and targets high performance computing market. The Tesla product has gone through the 10-serials, and the Fermi architecture (20-serials) and the new Kepler architecture. The Kepler-based GPU was first introduced in early 2012. Currently, only a few top supercomputers (e.g. Titan and Blue Waters) are installed with this new chipset, some other supercomputers are expected to be upgraded from Fermi to the Kepler or even newer architectures in 2014.

The NVIDIA Fermi GPU was the most widely used product for GPU developers up to 2013. Its fastest product is "Tesla M2090", which was released in May 2011. As shown in Figure 2.3, the Tesla M2090 has 16 Streaming Multi-Processor (SM) and 512 CUDA parallel processing cores in total (each SM contains 32 CUDA cores, running at 1.3 GHz). It delivers 1,331 GFLOPS in single-precision (SP) performance and 665 GFLOPS in double-precision (DP), with 6GB GDDR5 memory and 177GB/sec memory bandwidth [18].

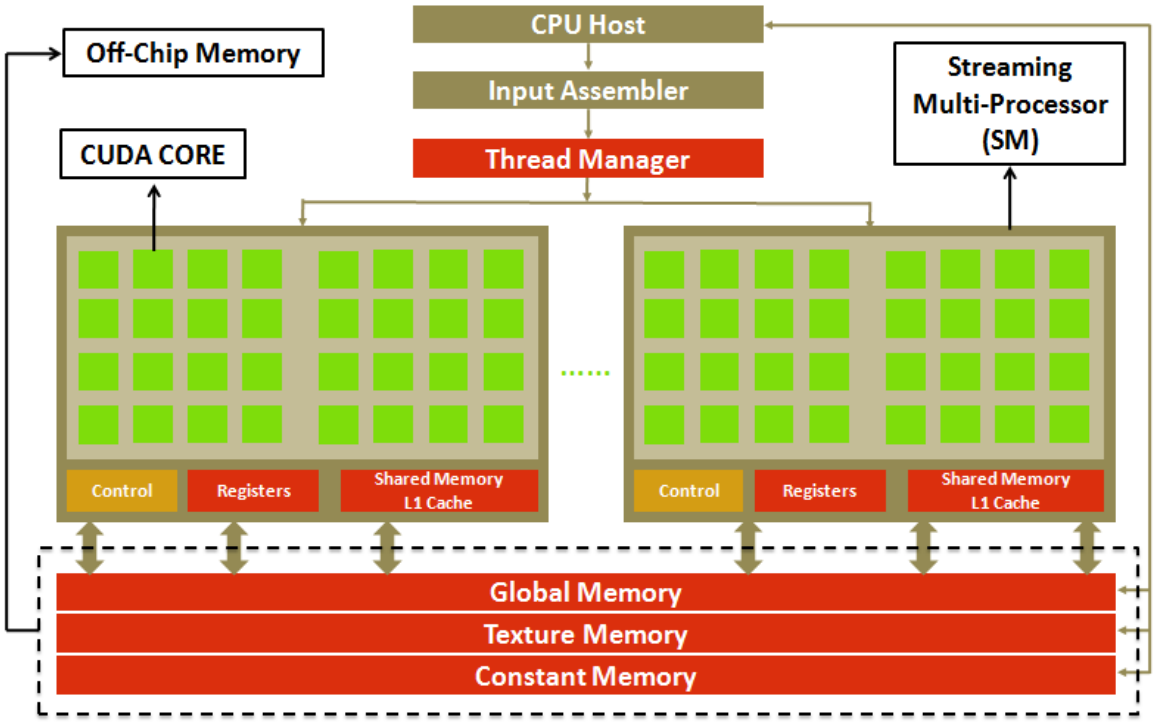


Figure 2.3: Abstract model of the NVIDIA Tesla M2090 GPU device controlled by a general purpose CPU [18].

The memory hierarchy of the Tesla M2090 includes high latency off-chip memory (global device memory) and low latency on-chip memory (private registers, shared memory, L1 Cache). The 6 GB global device memory on Tesla M2090 is shared by all 512 CUDA cores. The 64 KB local SRAM (Static Random Access Memory), including software-managed shared memory and hardware-managed L1 cache, can only be accessed by the 32 CUDA cores inside the same SM. The private registers (maximum 63) are specific to a thread but not visible to any other threads, even in the same thread group. One important feature for the Fermi architecture is that the 64KB local SRAM can be split into two different modes: 16KB for L1 and 48KB for shared memory or 48KB for L1 and 16KB for shared memory, which provides considerable flexibility and optimization space for kernel design on the GPU. All other Fermi GPUs have the same architecture as the Tesla M2090 but with different numbers of CUDA cores or global memory size.

In general, Kepler is an architecture that is quite similar to Fermi. They have the same execution model, local SRAM feature, warp size and etc. However, Kepler provides more advanced features. We'll use the NVIDIA Tesla K20 as an example, which is GK110 Kepler architecture based and widely installed in top supercomputers. Each SMX in Tesla K20 contains six times the CUDA cores in the Tesla M2090 (192 cores vs. 32 cores). The clock frequency of each CUDA core in K20 has been lowered due to power dissipation limits, but the computing performance delivered by K20 is still three times better than the Tesla M2090. Secondly, each SMX has three schedulers instead of one in the SM of the Tesla M2090, incorporating two important features: Dynamic Parallelism and Hyper-Q. Dynamic Parallelism allows GPU kernel to launch its child kernels without going back to the CPU, while in Fermi, all GPU kernel launches must be controlled the CPU, which creates a lot of communication overhead between the CPU and GPU if there are father-son kernels. Hyper-Q provides more flexibility to GPU programmers, which makes sure multiple CPU cores can submit and run GPU kernels/tasks on a single GPU simultaneously. The Hyper-Q in K20 provides a maximum of 32 hardware-managed connections between the host CPU and the GPU, which strongly supports the hybrid CPU/GPU programming [19].

When executing kernels on the GPU, the CPU controller must transfer the data to the GPU global device memory via the PCIe and then invoke the GPU hardware for parallel computation. The GPU kernel runs a set of thread groups, with each thread group assigned to a

SM and each thread assigned to a CUDA core. Each CUDA core has an individual ALU (Arithmetic Logic Unit) and all CUDA cores in the same SM execute the same instruction. Therefore, branches and thread divergence should be avoided within the same thread group, otherwise it will cause threads to be executed in serial and kill the parallelism. Although each SM only contains 32 CUDA cores, the size of each thread group can be up to 1,024 or larger. Each thread group is divided into subgroups of 32 and all subgroups running on the same SM are arranged and managed by the GPU scheduler.

Due to the limitation of on-chip memory, GPU programmers should find an appropriate size for the thread group in order to achieve good device occupancy. Device occupancy means the ratio of the number of active subgroups to the maximum supported subgroups on a single SM. High occupancy can always help to hide the data transfer latency from global memory to the on-chip memory. For example, if there are two active subgroups on the SM, the hardware will do the computation for the first subgroup and also fetch the data prepared for the second subgroup simultaneously. However, higher occupancy does not guarantee better computation performance. Computing performance not only depends on occupancy, but also combined optimization methodology.

Table 2.1 is a hardware specification comparison between Intel i-7 9600 CPU and NVIDIA Tesla M2090, showing the NVDIA GPU has significantly better computing capability than the CPU. Fundamentally, the CPU can provide the best performance for single thread tasks, but the limited number of cores restricts the data parallelism in the CPU. The GPU sacrifices single-thread computing performance and some general purpose processing capability (e.g. branch prediction), but provides more parallel processing cores and higher parallel efficiency for large scale throughput computing.

Table 2.1: Hardware Specification comparison between Intel i-7 9600 CPU and NVIDIA Tesla M2090

	Num. of Processor Element	Frequency (GHz)	SP SIMD width	Memory Bandwidth (GB/sec)	Peak SP SIMD GFLOPS
Intel i-7 9600	4	3.2	4	32	102.4
NVIDIA M2090	16	1.3	32	177	1,331

To compete with NVIDIA in the HPC market, Intel announced a new product called the Many Integrated Core (MIC) architecture in June 2011, aiming to enter into the era of HPC exascale computing solutions. The goal of the Intel MIC design is to create a x86-compatible many-core processor. Current software can benefit from the MIC architecture directly without changing any existing parallelization software tools [20]. The first product based on Intel MIC targeting HPC is codenamed "Knight Corner", building on a 22-nanometer manufacturing process and featuring the world's first 3-D Tri-Gate transistors. It will scale to more than 50 Intel processing cores on a single chip [21]. Based on the current top 500 supercomputers released in Nov. 2013, the No.1 supercomputer China's Tianhe-2, is powered with this new Intel MIC architecture "Xeon Phi", which has two Intel Xeon IvyBridge processors and three Xeon Phi processors with more than 190 cores in each computing node [22]. The new "Stampede" upgraded from "Ranger" at Texas Advanced Computing Center (TACC) also includes Intel Xeon Phi co-processors and provides more than 7 PetaFlops computing capability [23]. It is widely expected that more supercomputers will be designed with the MIC architecture in the next 1 or 2 years to compete with the current hybrid CPU and GPU mode.

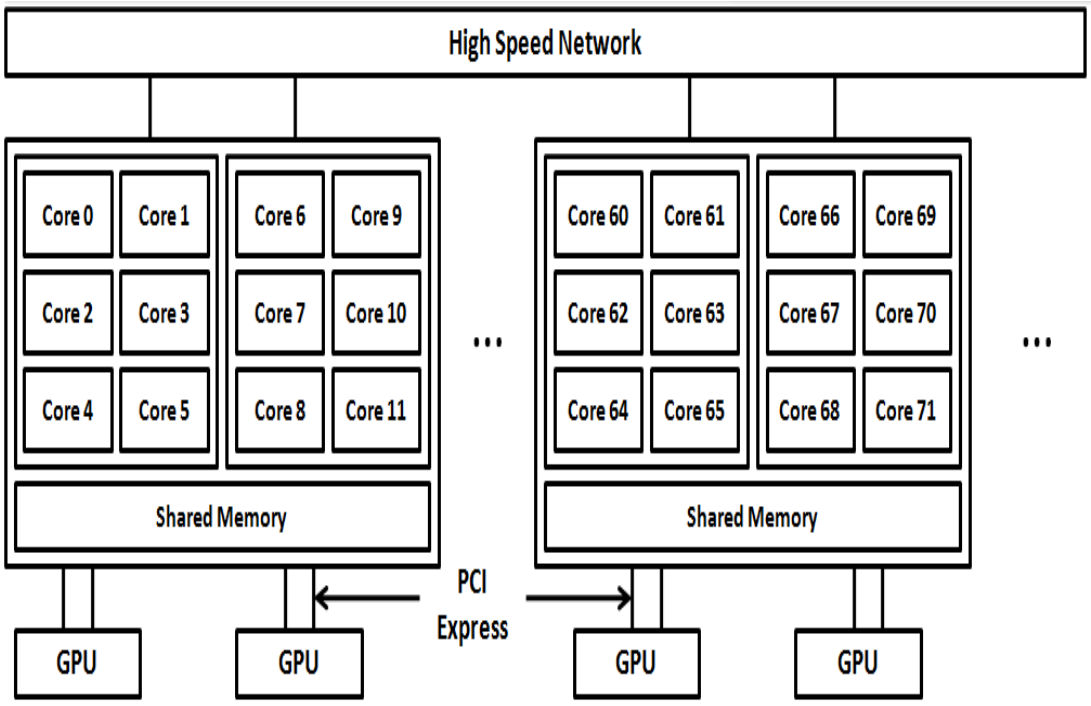


Figure 2.4: Abstract model of the heterogeneous supercomputer for OLCF Titan and NICS Keeneland. The significant difference between the two machines is the number of CPUs and GPUs in each node and the interconnections between nodes.

Looking back, the first PetaFlops heterogeneous supercomputers powered with the combination of a CPU and GPUs is the Tianhe-1A machine in Tianjin, China released in 2012. Our research work is primarily based on more advanced heterogeneous supercomputer OLCF Titan (No. 2 of Top 500 released in Nov. 2013), and also the NICS Keeneland machine for development (Figure 2.4). The OLCF Titan is the Oak Ridge Leadership Computing Facility's (OLCF) next generation, leadership-class supercomputers based on the Cray XK7 hybrid architecture [17], which is upgraded from the original Jaguar system, based on the Cray XK6 system [17]. Currently this giant machine provides a theoretical peak performance of 27.1 PetaFlops and consists of 18,688 computing nodes. Each computing node in OLCF Titan is equipped with one AMD' 16 cores Opteron^{TM} 6200 serial processor, and one NVIDIA Tesla K20 Kepler GPU accelerator, connected by a PCI Express GEN2 interface. The interconnection between nodes is Gemini, which is 3-dimensional torus topology and provides over 20GB/sec bandwidth per node. The Georgia Institute of Technology Keeneland machine is located at NICS, which includes 120 computing nodes with total peak performance 255 TeraFlops. Each node of this small supercomputer contains two Intel Westmere hex-core CPUs and 3 NVIDIA Fermi M2090 GPUs, with a total of 120 nodes, 240 CPUs and 360 GPUs. The interconnection is Qlogic QDR InfiniBand [24].

Peak performance has been mentioned often in previous sections when introducing supercomputers. Generally, peak performance is just theoretical number and we can easily calculate it using the followed formula:

$$CPU\_FLOPs = Core\_Num * Core\_Frequency * SIMD\_width \quad (2.1)$$

$$GPU\_FLOPs = Core\_Num * Core\_Frequency * 2 \quad (2.2)$$

$$Total\_FLOPs = (CPU\_FLOPs + GPU\_FLOPs) * Num\_Compute\_Node \quad (2.3)$$

In Equation 2-1, the SIMD_width is 4 for AMD CPUs and 8 for Intel CPUs due to the architecture difference. In Equation 2-2, the MAD or Multiply-ADD instruction has been counted as 2 FP operations/per clock cycle in the GPU, so the peak performance is doubled because all operations could be MAD.

To validate the numbers published in Top500 supercomputers, we use OLCF Titan as an example. Each compute node in OLCF Titan includes 16-AMD CPU cores and 1 Tesla K20x GPU. The peak performance of the AMD 16-core CPU is 16 cores * 2.2GHz * 4 SIMD = 0.1408

TFLOPs, and the peak performance of NVIDIA Tesla K20x GPU is 896 DP units x 0.732 GHz * 2 = 1.31 TFLOPs. Since the total number of compute nodes is 18,688 on the OLCF Titan, so the peak performance in total is (1.31 + 0.1408) * 18,688 = 27.1125 TFLOPs, which exactly matches the number published by the Top 500 [1].

However, peak performance is just a theoretical value, and none of application codes in the world can achieve that level, because the data fetch always take time even if the cache hit rate is 100%. Also not every operation is FP (e.g. branch) or can be completed in single clock cycle (e.g. division). Furthermore, most operations in the GPU program are Non-MAD instead of MAD. Computation pattern is largely determined by the algorithm. For this reason, a programmer cannot replace all computation with the MAD instructions. Therefore, the peak performance for Non-MAD GPU applications is considered to be up to 50% of the Full-MAD ones.

LINPACK software has always been utilized to measure the actual performance of supercomputers. The LINPACK uses the BLAS (Basic Linear Algebra Subprograms) to perform vector and matrix operations [25]. The algorithm is highly computation intensive, and MAD friendly for GPU as well. The Top 500 rank is based on the outcome of LINPACK and updated every six months. In the latest list released in Nov. 2013, the maximum performance measured and the theoretical peak performance for the No.1 China's Tianhe-2 machine are 33.8 PFLOPs and 54.9 PFLOPs respectively. And the maximum performance measured and the theoretical peak performance for the largest CPU-GPU heterogeneous supercomputer OLCF Titan are 17.5 PFLOPs and 27.1 PFLOPs respectively.

2.2 Parallel Programming Tools and Models

Programming on heterogeneous supercomputers must deal with multi-core general purpose CPUs and multi-core modern GPUs. Applications designed for single-thread CPUs must be completely restructured and redesigned to make full use of the massive parallelism provided by supercomputers. Many parallel programming tools and models targeting HPC on supercomputers can help programmers control the parallel computing units and complex memory hierarchy. Understanding these parallel programming tools and models can help programmers understand the underlying architecture better and achieve higher computation performance. We will present the most popular parallel programming tools and models for heterogeneous

computing in this section.

2.2.1 OpenMP (Open Multi-Processing)

OpenMP is a parallel programming model for shared memory architectures and also the industry standard API for multithreaded programming on shared memory architectures [26]. For shared memory architectures, a number of CPU processors/cores share the same physical memory space. If these processors are utilizing a single address space and have the same view of the physical memory, communication can be easily implemented and controlled through shared data stored in the memory. Atomic operations or semaphores are also supported in shared memory architectures to avoid conflict when accessing the shared data. Based on these key features of the shared memory architectures, OpenMP has been released to help programmers write multithread applications simply and efficiently. The first version of OpenMP was released in 1997, which only supported the Fortran language. The latest version, 4.0, was released in July 2013 and supports C/C++ and Fortran on most processing architectures and operating systems [27].

OpenMP programming is just a set of high level compiler directives for parallel application programmers. The OpenMP-enabled compiler automatically generates parallel codes based on the annotation defined in the program. The OpenMP programming model is based on Fork-Join parallelism. The program executes a single master thread, that will spawn a team of threads when encountering annotated parallel codes. All multi-threading execution is invisible to programmers but implemented by the compiler. Figure 2.5 is an example for the loop work sharing use case.

```
void main()
{
    int i = 0, Max = 64, A[100];
    #paragma omp parallel private (i) shared (A, Max)
    for(int i=0; i<Max; i++)        A[i] = i;
}
```

Figure 2.5: An OpenMP example for loop work sharing use case. The compiler will divide the iteration space into a lot of small chunks. Each thread will be executing on the same loop code but only on its own chunk. Suppose there are 8 threads requested, then the thread No. “N” will process variable “i” ranging from N*8 ~ N*8 + 7. Variable “i” is different and variables “A” and “Max” are same for each thread, so variable “i” is defined as private while variables “A” and “Max” are defined as shared.

OpenMP is a relatively easy to implement. However, the shared memory programming model has some scalability issues. The memory bus becomes the performance bottleneck if more and more processors concurrently access the same physical memory. In addition, programmers cannot control the parallelism, which might not achieve the best performance because of overhead caused by the compiler.

2.2.2 MPI (Message Passing Interface)

MPI is a parallel programming model for distributed memory architecture. Unlike the shared memory architecture, each core has its own memory. Even though some processors might share the physical memory, the memory space for each processor is still invisible to others. The biggest advantage of the distributed memory architecture is scalability. The number of processors in a machine can be as large as possible. The biggest disadvantage of the distributed memory architecture is data communication. All data communication between processors needs to pass through the interconnection (shared physical memory or network), and the scalability of a machine is strongly limited by the network capability. In addition, programmers must take care of the data communication instead of the compiler in OpenMP.

MPI is an industry tool and standard for data communication on distributed memory architectures. It is language independent and supports both point-to-point and collective data communication [28]. The first MPI version was released in 1994 after a draft version discussed and presented at the Supercomputer Conference in 1993 (SC'93). The latest and most popular MPI version, 1.3, was released in 2008. The MPI-2, which includes many new features such as parallel I/O, dynamic process management, remote memory control, is a subset of the MPI version 1.3 [28].

MPI provides a very strong capability to help programmers develop parallel applications on large machines. However, extra effort or algorithm designs are still required to achieve better performance when using MPI. For example, programmers can choose either a blocking or non-blocking style of point-to-point communication. Blocking helps to guarantee the shared data is 100% correct before processing the next step, but the communication latency might be high. Non-blocking can reduce the data communication latency effect, since CPUs can do some computation work while waiting for new shared data, but programmers need to check the correctness of the data before the CPU does any computation.

2.2.3 CUDA (Compute Unified Device Architecture)

CUDA is a parallel programming model designed by NVIDIA, running on NVIDIA GPU devices for general purpose processing. The CUDA programming model provides both low level and high level APIs to support the C/C++ and Fortran languages. All operating systems running on the desktop with NVIDIA GPUs and the Android system running on NVIDIA Tegra chipset, supports CUDA programming. The first CUDA SDK was released in 2007, and has been updated every year since then. Currently the latest CUDA SDK is version 6.0. The new CUDA 6 offers many new features to make GPU programming simple and provide high performance [29]. NVCC is the NVIDIA CUDA compiler and compiles both host (CPU) and device (GPU) code.

The basic unit for CUDA programming is the thread. Each thread has a global ID and local ID for programmers to assign its computation work. Unlike the CPU hardware thread, which has a private program counter (PC) and works independently, CUDA threads are grouped into 32 as a warp. A CUDA thread is mapped into the real CUDA core in the NVIDIA GPU hardware. All 32 threads in the same warp share the same PC and follow the same execution. For branches or diverges, threads in the same warp have to be serialized to finish the execution, which will cause a performance penalty because some threads are idle instead of computing.

A GPU kernel requests thousands of hundreds of CUDA threads to execute the computation. All threads are divided into blocks and blocks are mapped onto the Streaming Multi-processor (SM) in order. For example, the NVIDIA M2090 has 14 SMs, so only 14 blocks can be running simultaneously. Once one block is finished with its computation, the next available block will take over the resource. Inside each block, warps scheduling and management are controlled by the hardware scheduler. As shown in Figure 2.6, the GPU scheduler helps to hide the data access latency and keep all CUDA cores busy on computation. In addition to sharing the computing resource, threads/warps in the same block can also share data through shared memory. However, synchronization is required when using shared memory, to guarantee the shared data is updated before use. GPU programmers must be careful in determining where to put the variables. Private registers are fastest but limited in number and cannot be shared. Shared memory is larger, fast but needs synchronization. Global memory is largest and shared, but the access latency would kill the performance.

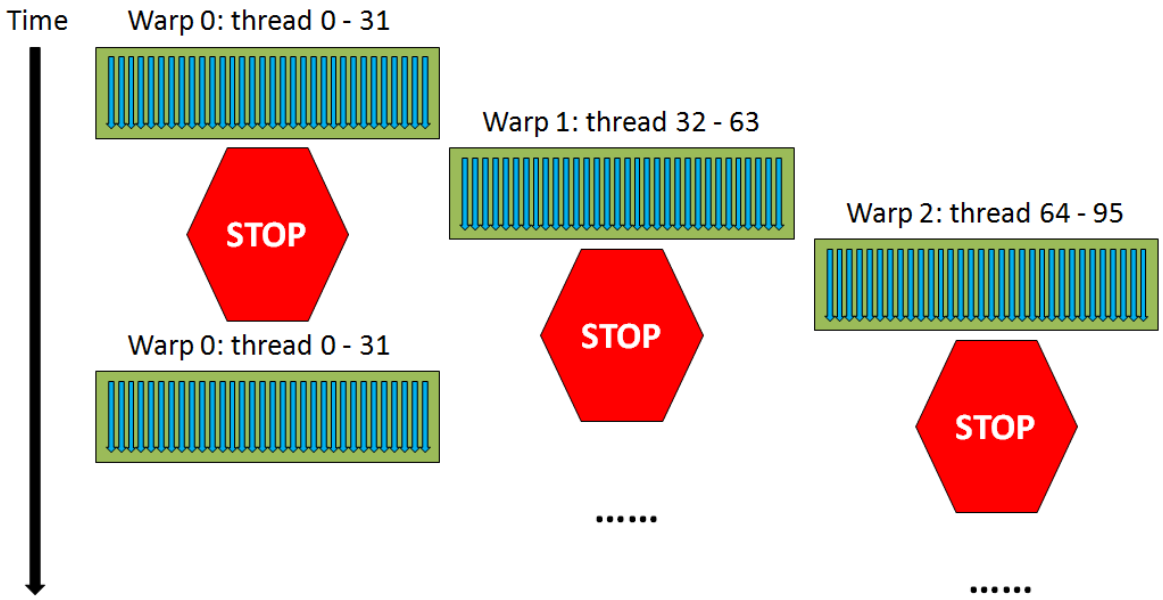


Figure 2.6: Warps scheduled and managed by the GPU scheduler (suppose occupancy is very low and only a single warp can be mapped to the SM). First, warp 0 takes over the hardware to do the computation. Once warp 0 starts to perform data access (fetching or storing data from global/shared memory), the scheduler switches to warp 1 and lets warp 1 take over the hardware resource and start the computation. When warp 1 starts the same data access and warp 0 has not finished the data access yet, the scheduler continues to warp 2 and keeps the GPU busy on computation. After warp 2 enters into the data access mode and warp 0 is done with the data access, the scheduler will switch back to let warp 0 resume its computation work.

2.2.4 OpenCL (Open Computing Language)

OpenCL is an industry open parallel programming standard for heterogeneous computing systems including CPU, GPU, DSP, FPGA and other processors (see Figure 2.7 for details). OpenCL standard is maintained by non-profit technology consortium Khronos Group [30]. The OpenCL was initially developed and supported by Apple only, and now it has been adopted by most of the top industry vendors including: Apple, Intel, Qualcomm, AMD, NVIDIA, Samsung, ARM Holdings and some others [31]. Some companies developed their own similar languages instead of supporting OpenCL, such as Renderscript from Google [32] and Directcompute/C++ AMP from Microsoft [33]. The first OpenCL 1.0 was released in late 2008 and the latest version is OpenCL 2.0 released in July 2013. OpenCL 2.0 includes a lot of new programming features and provides more flexibility to programmers, such as shared virtual memory (SVM) and Dynamic parallelism.

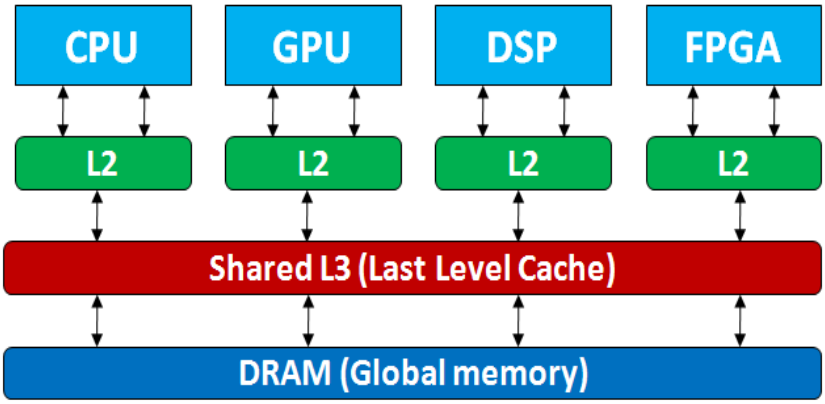


Figure 2.7: Ideal heterogeneous system architecture (HSA) for OpenCL programming: using a single programming language to control all four different computing devices in the same application. Each computing device has its private L2 cache. Fast data sharing via Shared L3 instead of DRAM can minimize data communication latency between different computing devices.

The OpenCL programming model is quite similar to CUDA. The host device launches kernels on the computing device, and each kernel requests thousands of threads to finish computing tasks in parallel. There is some different terminology between CUDA and OpenCL including: “thread” vs. “work-item”, “block” vs. “work-group”, “shared memory” vs. “local memory” “warp” vs. “wave” and “streaming multi-processor” vs. “shader processor”. In addition, CUDA supports more features than OpenCL, such as the Hyper-Q, which allows multiple CPU cores to launch and run their kernels on the same GPU simultaneously. However, CUDA is limited to the NVIDIA GPU product, while OpenCL is supported by most computing devices with great portability. To compile OpenCL code, a company always has its own compiler support, such as the AMD parallel processing software kit and the Qualcomm LLVM (Low Level Virtual Machine) compiler. OpenCL code can be compiled during run-time or pre-compiled as binary like the pre-compiled CUDA code by the NVCC.

The NVIDIA GPU also supports OpenCL. However, the performance gap between CUDA and OpenCL is large when running an identical code on the same NVIDIA GPU device. Fang et. al. [34] did a comprehensive performance comparison of CUDA and OpenCL, and they found CUDA performs at most 30% better than the OpenCL. The reason could be CUDA is specificly designed for NVIDIA GPUs and can make full use of the hardware for optimization and tuning. But OpenCL must support all kinds of computing devices and has to sacrifice some optimization options due to the architectural differences.

2.2.5 Future Parallel Programming model - OpenACC

OpenACC is a parallel programming model newly proposed by Cray, CAPS, NVIDIA and PGI [35]. Similar to OpenMP, the OpenACC offloads loops or parallel parts from the host CPU to attached accelerators (GPU, Cell processors, etc) for high performance computation. The programmer inserts directives before the loop in their high-level language program, then the OpenACC compiler and run-time libraries help to manage all computing tasks on accelerators, including initiating and releasing devices, and data transfer between devices. Programmers do not have to learn the architecture of the underlying computing system, but simply provide the performance-related information to the OpenACC compiler. The main purpose of the OpenACC is to simplify parallel programming on heterogeneous computing platforms. The OpenACC was first introduced at the International Supercomputing Conference 2012 (ISC'12) and the draft version was presented at Supercomputing 2012 (SC'12). In 2013, the commercial compiler PGI began supporting OpenACC and the GCC compiler is also adding support for OpenACC [36].

Because OpenACC is not widely utilized yet, there are limited publications about the performance comparison between OpenACC and manually optimized codes. In OpenACC programs, only regions indicated by programmers run on the accelerators, while other parts execute on the host CPU. There might be more interactions between host and accelerators, bringing larger overhead than the manually optimized codes. In addition, the OpenACC compiler might request more register usage than manually optimized code, which would reduce the parallel efficiency and could not maximize the computation performance. Nevertheless, as the computing hardware continues upgrading and compilers keep improving, we believe OpenACC will become more powerful in the near future, comparable to CUDA and OpenCL as a main tool for parallel programming today.

2.3 Petascale Stencil Computation Applications

As the computation capability of top supercomputers has scaled from terabyte to petabyte levels in last five years, many large scale scientific applications have benefited from this increased computing power and achieved significant performance gains after aggressive tuning. Here we briefly show some world famous scientific computation applications, that have some similarity to our AWP-ODC applications in either the use of stencil or in application. All

application information and result numbers are from their publications. More details can be read through reference papers [2, 37-46].

2.3.1 Phase-field Simulation

The Gordon Bell Prize winner at SC'2011 by T. Shimokawabe et al. from Tokyo Institute of Technology demonstrated a multi-GPU implementation of phase-field simulation modeling for dendritic solidification, that achieved 1.017 PetaFlops in single precision using 4,000 GPUs and 16,000 CPUs on TSUBAME 2.0 Supercomputer [2].

The phase-field model introduces a continuous parameter to describe whether the material is solid or liquid. The numerical model includes one second-order finite difference scheme for space and one first-order forward Euler-type finite difference method for time on a 3D regular computation grid (shown in Figure 2.8a). To compute a center point (i, j, k), a total of 26 elements (6 phase field values and 19 concentration values) need to be read from memory, and 2 elements (one phase field value and one concentration value) written back to memory.

TSUBAME 2.0 supercomputer is located at Tokyo Institute of Technology, and is now upgraded to TSUBAME 2.5 ranked as No. 11 based on Top 500 released in Nov. 2013. TSUBAME 2.0 contains 1,408 computing nodes, where each node has two 6-core Intel Xeon X5670 CPUs and three NVIDIA Tesla M2050 Fermi GPUs. The interconnection is fat-tree with 200 Tbps bi-section bandwidth. The full GPU simulation code is compiled by CUDA version 3.2.

T. Shimokawabe et al. have developed three multi-GPU implementations in their search for higher computation performance, including a GPU-only method, a Hybrid-YZ method and a Hybrid-Y method. The GPU-only method allocates each individual sub-domain into a separate GPU, and the GPU takes over the entire computation. For the Hybrid-YZ method, the CPU computes outer region, which contains all data to be swapped, and the GPU computes the rest inner region. The Hybrid-Y method balances the load between CPU and GPU, meaning computation time and communication time spent by CPU and GPU are almost equal. As shown in Figure 2.8b, the weak scaling result shows the Hybrid-Y method achieved both high computation performance and better scalability.

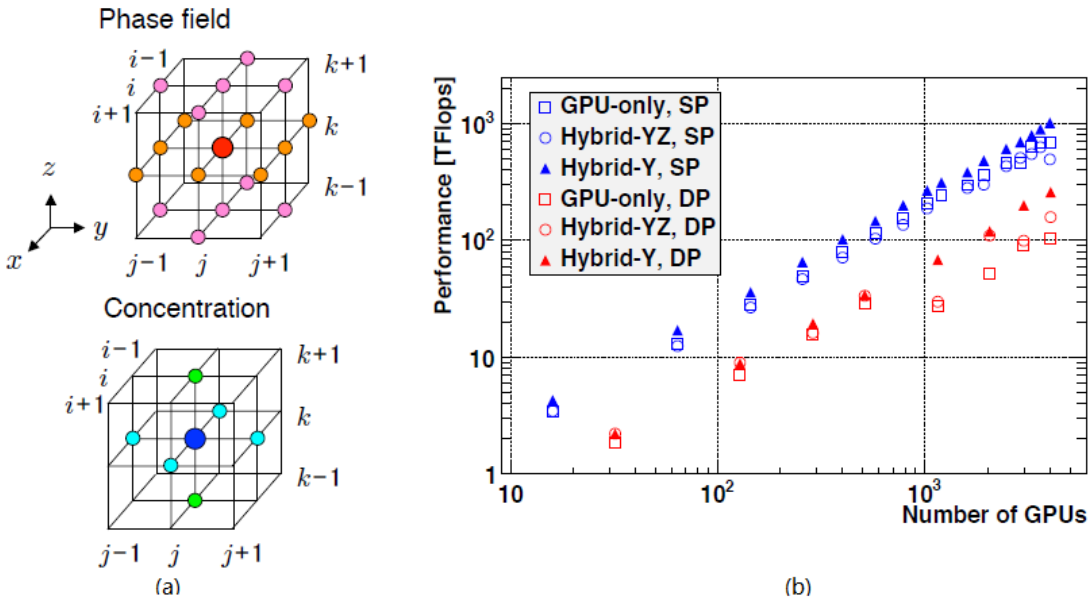


Figure 2.8: (a) Spatial access pattern for the phase field and concentration. (b) Weak scaling performance by using a GPU-only method, a Hybrid-YZ method and a Hybrid-Y method in both single precision (SP) and double precision (DP) [2].

2.3.2 Global Atmospheric Simulation

Global atmospheric simulation is a key component in climate modeling and helps scientists understand dynamic behaviors of the global atmosphere. Yang et al. from the Chinese Academy of Science developed a peta-scalable hybrid algorithm applied in a cubed-sphere shallow-water model for global atmospheric simulations. Their experiments demonstrated nearly ideal strong and weak scaling on Tianhe-1A supercomputers, with sustained performance of 0.8 PetaFlops in DP (1.6 PetaFlops in SP) using 45,000 CPU cores and 3,750 GPUs [37].

The numerical model of the shallow-water equations has been simplified into two 2D second order finite difference schemes, which requires two 2D stencil computations at each time step. For each cell (i, j), 13-point mesh cell information is read from memory and the updated cell writes back to the memory (shown in Figure 2.9a). Yang et al. developed CPU-only and hybrid CPU+GPU algorithms. The main idea is to balance computational load and reduce the communication latency between the CPU and GPU. As shown in Figure 2.9b, the tuned CPU+GPU method has achieved nearly perfect weak scaling on Tianhe-1A as the number of computing nodes increased from 6 to 3,500.

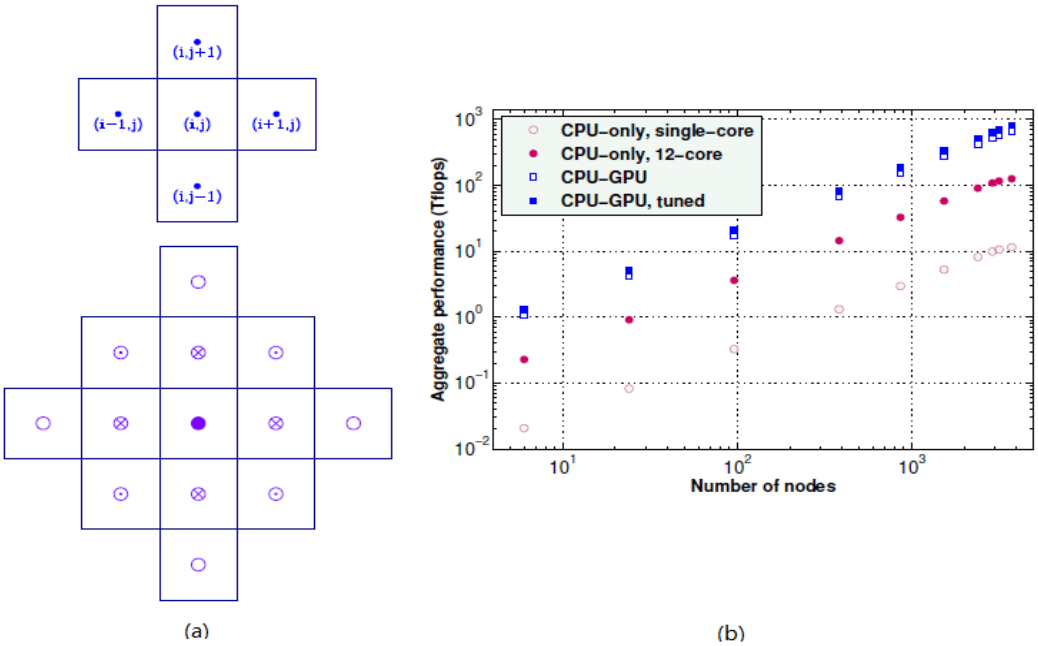


Figure 2.9: (a) Top: state reconstruction in cell(i, j). Bottom: 13-point stencil exhitbits a diamond shape. (b) Weak scaling results on Tianhe-1A using CPU-only and hybrid CPU+GPU [37].

2.3.3 Seismic Simulation

Seismic simulation is to quantify earthquake hazards in physics-based models that help to understand risk and improve resilience in seismically active regions. Detailed simulation hazard maps can provide system engineers, emergency responders and disaster planner with a realistic, high resolution scenario to understand the earthquake damage. In 2010, the largest ever wall-to-wall earthquake simulation on San Andreas Fault was achieved on OLCF Jaguar CPU-based machine, with the seismic frequency exceeding 2Hz and the computation size approaching 10^{17} mesh points [5]. The arrival of new heterogeneous supercomputers has opened the door to even higher frequencies and spatial resolution, and it provides seismologist more opportunities to simulate and study earthquakes at petascale level for the first time.

A number of previous research projects focused on acceleration of seismic simulations have used a heterogeneous computation solution (CPU + GPU) [38-46]. Dimitri Komatitsch and his research team at CNRS/University of Aix-Marseille France has implemented the first GPU version of seismic simulation code including SPECFEM3D [38], finite-difference time domain (FDTD) [39] and finite-element [40-41]. Komatitsch et al. scaled their seismic simulation codes up to 192 Tesla 10 serial GPUs and achieved more than a 20 times speedup compared to

CPU-only codes. Abdelkhalek et al. [42] demonstrated 10 times and 30 times speedups with their 3D FD implementation of reverse time migration (RTM) and seismic application respectively on 8 NVIDIA 10 serial GPUs. Song et al. [43] accelerated the Support Operator Rupture Dynamic (SORD) code with 14 times speedup on 64 NVIDIA 10 serial GPUs. Okamato et al. [44-46] successfully accomplished the Tohoku-Oki earthquake simulation using a CPU/GPU solution, and reached 2.2 TFlops on 120 GPUs of the TSUBAME supercomputer.

2.4 Summary

In this chapter, we described two kinds of modern supercomputer architectures in details, including homogeneous and heterogeneous. General CPUs plus acceleration units have been the main stream for heterogeneous computing nodes, with CPU+GPU as the most popular one based on Top500 supercomputers released in November 2013. As the computation performance primarily relies on massive on-chip parallelism controlled by both hardware and software, programmers must better understand the architectures, and become familiar with programming models in order to deliver high performance solution. MPI+CUDA has been selected as our programming model and tool. This is because we target OLCF Titan supercomputer, in which each computing node consists of one AMD 16-core CPUs and one NVIDIA Kepler GPUs.

We also introduced some of the advanced petascale stencil applications including seismic codes using heterogeneous solution. Compared to our AWP code, two petascale applications have smaller order of finite difference methods (4 vs. 2) and also deal less with memory-bound issue. None of the current seismic simulation codes mentioned in Section 2.3.3 have been extended to petascale level. These research investigates accelerations and time-to-solution reduction using smaller CPU/GPU clusters compared to CPU-only solutions. In the following chapters, we present new AWP code with nearly perfect scalability achieved. The new AWP provides seismic scientists, for the first time, with petascale ability to simulate ground motions from large fault ruptures to frequencies as high as 10 Hz in a physically realistic way.

Chapter 3

Earthquake Simulation Workflow

This chapter presents the whole earthquake simulation framework and the scientific workflow designed for the petascale earthquake simulation running on the XSEDE. Generally, earthquake simulation involves significant bookkeeping. This includes data collection for the simulated earthquake regions, translating these data into the simulation format, pre-processing these data before the simulation, running the full simulation on supercomputers and post-processing data to generate the hazard mapping for scientific research or industrial purposes. Our role in the simulation is to design the simulation and pre/post-processing software, run SCEC seismic simulations on XSEDE. SCEC provides the input meta-data. The simulation output is visualized by Amit Chourasia at SDSC, and the results are analyzed by the SCEC community.

Moving from terascale to petascale, large scale earthquake simulations invovle multiple steps, and several hours of supercomputer computation to complete a single run. For example, the M8 simulation run on Jaguar machine in 2010, took around 24 hours and generated several peta-bytes of data for scientific research. Generally, it is unrealistic to repeat such a large capability simulation because of the allocation cost and significant manpower invovled. Moreover, there could be some data corruption during the peta-byte data transmission between supercomputers due to the network or file system issue, which would cause the whole simulation results useless and also a huge waste in computation hours. Therefore, we came up with an automatic workflow developed to support the data management of a petascale earthquake simulation, which helps manage the simulation process and make sure all data produced are valid and safely backed up.

In this chapter, Section 3.1 describes some project background and motivation, including the XSEDE (replaced and expanded from previous TeraGrid) platform and our project center SCEC. Earthquake components are briefly introduced in Section 3.2 in order to better understand why our scientific workflow is required. Section 3.3 presents the detailed scientific workflow, which includes the workflow framework, software tools and simulation results tested on XSEDE using the Shakeout simulation. Conclusions and future work will be presented in Section 3.4.

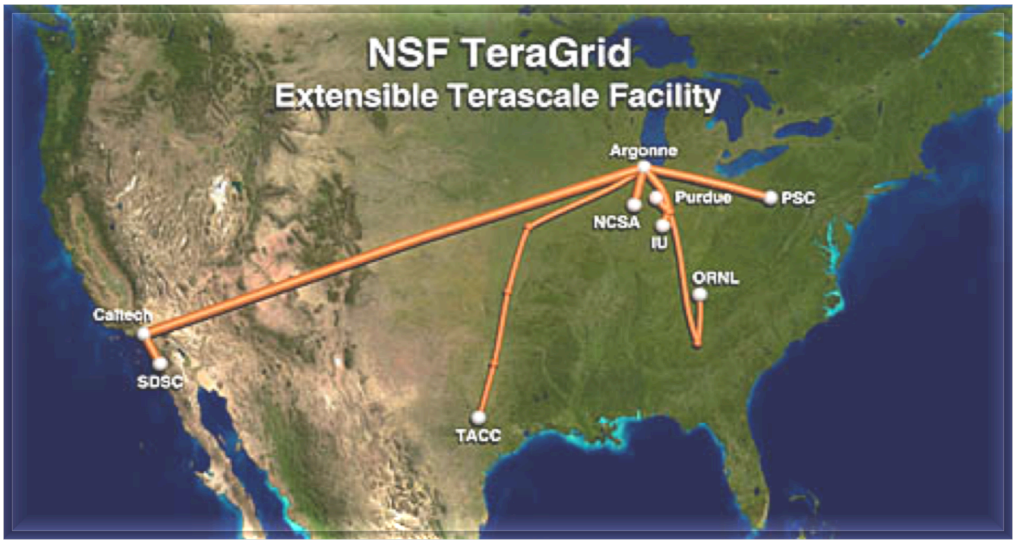


Figure 3.1: NSF XSEDE Facility, including nine major supercomputers located across the United States. San Diego Supercomputer Center (SDSC) is one of the main connector, so our research can make full use of the whole XSEDE computing resource. [47].

3.1 Background and Motivation

As shown in Figure 3.1, XSEDE of the United States is the world's largest most comprehensive distributed cyberinfrastructure for open scientific research. It includes almost 20 Petaflops of computing capability and 30 Petabytes for data storage and archiving. Recently the TeraGrid has been renamed as the Extreme Science and Engineering Discovery Environment (XSEDE) [48] and some supercomputers have also been upgraded to hybrid computing mode (CPU + GPU). Table 3.1 gives a system overview and technical specifications for the two top supercomputers (TACC-Stampede and NICS Kraken) in the XSEDE .

Table 3.1: System overview and technical specification for Stampede and Kraken [23, 49]

Supercomputers	TACC-Stampede	NICS-Kraken
Peak Performance	9.6 PetaFlops	1.17 PetaFlops
Compute Nodes	6,400 nodes and 522,080 cores	9,408 nodes and 112,896 cores
Memory	270 TeraByte	147 TeraByte
File System Storage	14 PetaByte	3.3 PetaByte
Inter-Connection	FDR InfiniBand Network	Cray SeaStar2+

Significant high performance scientific applications, such as those in bioinformatics, physics, earth science, astronomy, etc, are running on XSEDE. In recent years many of those grid applications have become more and more data intensive and massive amounts of data in such applications are shared and distributed across the XSEDE resources. Therefore, the ability to support large scale data transfer, management, distributed access and analysis becomes the key to adapting those data-intensive applications to exploit the aggregate capacity of the computing and storage resources on the XSEDE.

SCEC [50] is a large user of XSEDE, consuming more than tens of millions of allocation hours each year. Researchers in the SCEC Community Model Environment (CME) program have developed a geophyics and IT collaboratory platform that performs seismic hazard analysis and geophysical modeling in the Southern California area, including a Petascale Cyberfacility for Physics-based Seismic Hazard Analysis Research called PetaSHA. Now the PetaSHA simulations are moving from TeraScale to PetaScale and our research group, led by Prof. Yifeng Cui at SDSC, has developed an automatic end-to-end scientific workflow to support PetaSHA simulation on XSEDE using the AWP-Olsen-Day-Cui (AWP-ODC) code [51]. Figure 3.2 is the SCEC Shakeout wave propagation simulation output on the TACC-Ranger machine (the predecessor of TACC-Stampede) using the AWP-ODC code in 2009, which computed billions of mesh points and generated several hundreds of TeraBytes of data for a 1-Hz earthquake study on the southern San Andreas fault. As the largest mesh simulations move into hundreds of billions of elements range, dynamic source nodes on fault are reaching millions, decomposing files have increased to hundreds of thousands, terabytes of data are transferred between sites increasingly frequently. High performance data transfer and ingestion in this scientific workflow becomes essential to support the scientific analysis on the heterogeneous collection of computational and storage resources on XSEDE.

Based on the characteristics of data types in the scientific workflow and heterogeneous resources on XSEDE, we examine the issues related to performance optimizations of data transfer and ingestion in this scientific workflow to support SCEC PetaSHA simulations. Some popular tools for data transfer and management on XSEDE are introduced and invovled in our workflow, e.g. Globus and iRODs system. The primary goal is to ensure the correctness of data before running any computation jobs in the workflow and minimize the time-to-solution by reducing data transfer and archiving time.

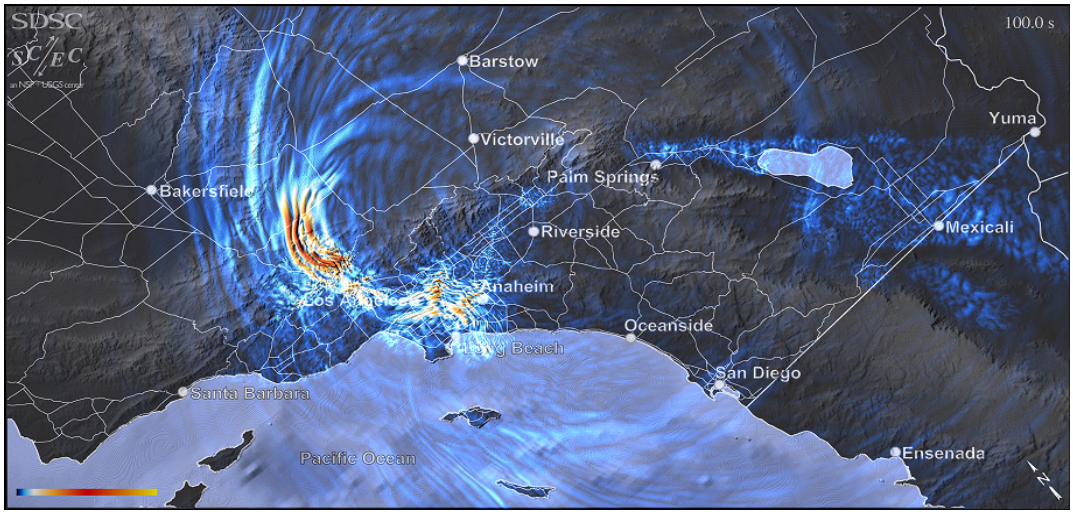


Figure 3.2: SCEC Shakeout simulation of a Mw7.8 earthquake on the southern San Andreas fault: 600 x 300 x 80 km domain, 100m resolution, 14.4 billion grids, upper frequency limit 1-Hz, 3 minutes, 50k timesteps, minimum surface velocity 500m/s, dynamic source, velocity properties SCEC CVM 4.0 model. This simulation will be used as an example case in Section 3.2/3.3 [52].

3.2 AWP-ODC Simulation Components

The AWP-ODC code is a fourth order finite difference 3D wave propagation code with a full package including pre-processing, simulation solver and post-processing. Here we briefly describe each component of the AWP-ODC software package in order to show why the scientific workflow is necessary for earthquake simulation. More details for each component of the AWP-ODC can be found in our SC'10 paper [5].

In earthquake simulation, the input data includes two types: mesh and source. The mesh data can be treated as the property of each point in the simulation domain and the source is the earthquake center that pumps energy out which propagates to other points. Generally, the number of mesh points equals the domain size, while the number of source points is limited and determined by the specific earthquake. The pre-processing tools in AWP-ODC have Mesh Generator CVM2MESH, Petascale Mesh Partitioner PetaMeshP, Dynamic Source Generator dSrcG and Dynamic Source Partitoner PetaSrcP. The simulation solvers in AWP-ODC are Dynamic Fault Rupture Solver DFR and Wave Propagation Model AWP. The post-processing is composed of Validation Toolkit aVal, derived Products dPDA and iRODs ingestion tool PIPUT. In the simulation solvers, there are also Mesh Reader and Source Reader designed to handle data distribution between different compute nodes/cores.

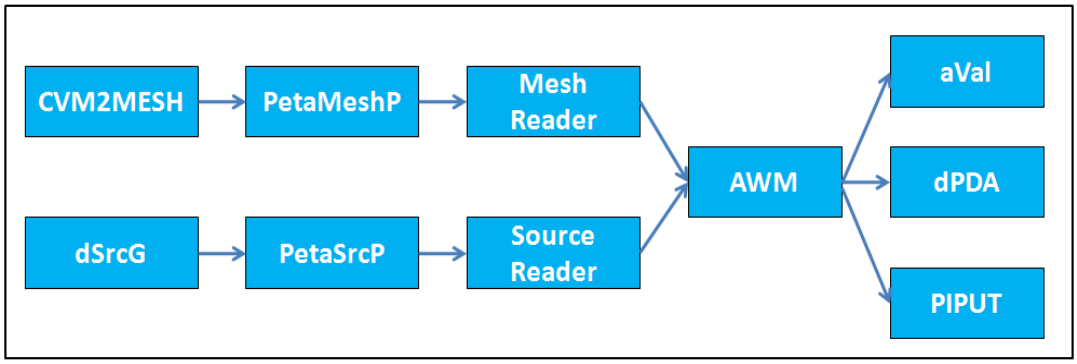


Figure 3.3: Components of the AWP-ODC, including input data pre-processing, simulation and output post-processing.

CVM2MESH is a mesh generator software developed by Patrick Small at SCEC, which is to extract material properties for each mesh point in the simulation domain. This software has been implemented using a scalable parallel algorithm. The program first partitions the 3D earthquake simulation domain into slices along the z-axis and then assigns an individual CPU core to extract the values of each mesh point in its slice from the original CVM model. The main computation is to find the corresponding position in the original CVM model for each mesh point, then set its values to the mesh point. In the end, all CPU cores write their outputs into a shared file created by the MPI-IO API for parallel writing. This mesh file will be saved as the input for the AWP-ODC earthquake simulation. Since the mesh file is for all mesh points in the 3D simulation domain, there is no need to store the location information. So the format for the mesh file is purely values saved in order, e.g. [values for 3D location (0, 0, 0)], [values for 3D location (0, 0, 1)], [values for 3D location (0, 0, 2)] …. [values for 3D location (N-1, N-1, N-1)]. This helps to maintain all the information with a minimum file size.

dSrcG is a Kinematic Source Generator tool to generate the moment rate time histories at a finite number of points in the simulation domain. Unlike the mesh file, the source file does not contain information for each mesh point, but only for a limited number of points. The format for each source file is the 3D location and the sequential full time serial moment information, e.g. [3D location], [timestep 1], [timestep 2] …. [timestep N]. The file size for the source not only depends on the number of source points, but also the frequency and the earthquake duration. If the frequency is high, then we will have more timesteps for the same amount of earthquake run time. Generally the file size is in the hundreds of giga bytes range due to the tens of thousands of source points and high frequency simulation.

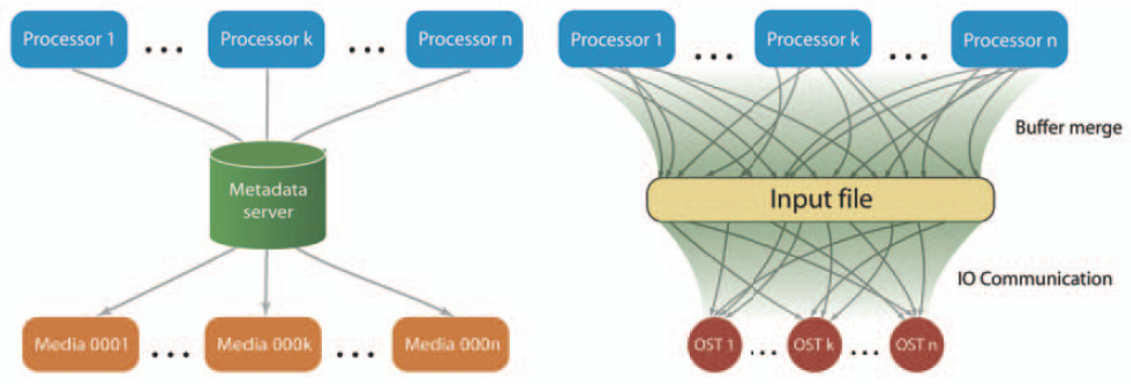


Figure 3.4: Two Partition Methods in PetaMeshP: the left is the serial approach and the right is the parallel approach [53].

CVM2MESH generates a single file of enormous size, in the hundreds of giga bytes or even several terabytes range, depending on the 3D domain size. Since the AWP utilizes hundreds of thousands of CPU cores to do the computation, each core can only handle a small amount of data due to the limited shared memory at each computing node, and PetaMeshP is the tool to pre-partition the mesh data before the main computation, which has provided the AWP scalable capability to handle different sizes of input mesh data.

PetaMeshP tool [53] includes a serial approach and a parallel approach, as shown in Figure 3.4. The serial approach is quite straightforward, and PetaMeshP generates the meta file for each CPU core. Suppose we request 1000 cores to do the computation, then the PetaMeshP will generate Media0001.bin, Media0002.bin, ….Media1000.bin. The number in the names of these partitioned mesh files corresponds to the rank of its CPU cores, for example, the rank 600 CPU core reads the Media0600.bin file to do the main computation. The advantage for this approach is easy implementation and it also reduces the load of the file system significantly, because each CPU just accesses its own file and no shared file access happens during file input. The disadvantage is that the mesh files must be regenerated to use a different number of CPU cores or decomposition topology. The parallel approach is designed to solve this disadvantage. Unlike the serial approach, the parallel approach does not generate these separate files, but directly passes the data into the right memory for each CPU core. The parallel approach has another academic name called “two layer data decomposition”. In the first layer, a small number of the CPU cores read big chunks of continuous data from the file system, and the second layer distributes the data into the right CPU cores. More details about the parallel approach are presented in SC’09 poster [53].

PetaSrcP is to distribute the source to the associated CPU cores. As discussed earlier, dSrcG generates a single moment-rate file, as the CVM2MESH does, and the source points only exist in a limited number. Therefore, PetaSrcP computes the right location for each source point and generate the source files for the associated CPU core. Since the data in the partitioned source file is primarily time serial information, we partition the source file again into smaller files based on time steps, which can help reduce the memory load for the CPU core. Otherwise, if there are several hundred source points belonging to the same CPU core and the simulation time is long, then the size of the partitioned file can be hundreds of mega bytes and not much memory space is left for computation. After the PetaSrcP data partition, the source files will be named: fault0001_0000.bin, fault0001_0300.bin, fault0001_0600.bin, …, fault0001_4800.bin, …, fault000N_0000.bin, fault000N_0300.bin, fault000N_0600.bin, …, fault000N_4800.bin. The first four digits mean the associated rank of CPU core and the last four digits are the beginning timestep of this source file. PetaSrcP has the same limitation as the PetaMeshP serial approach. We need to regenerate the partitioned source file if using different number of CPU cores or decomposition topology.

Mesh Reader is the beginning of the simulation, which has four options in the AWP-ODC. The first option is homogeneous mesh input, where the mesh property is the same for all mesh points and calculated based on some input parameters. The second option is to handle the smaller mesh/simulation size. Each CPU core computes its data offset and acquires the data directly from the same mesh file via MPI IO and the parallel file system. The third option is to read the partitioned mesh files generated by the PetaMeshP serial approach. This option is utilized when dealing with hundreds of thousands of processor cores to ensure scalable and high performance I/O bandwidth. The last option is to use the PetaMeshP parallel approach to read large mesh file directly.

Source Reader runs throughout simulation. After each timestep, the simulation must add new source time history information to simulation prepared for the next timestep. Source Reader consists of two options. The first option is to handle smaller source input. The main CPU core reads in all source information and then broadcasts it to all other CPU cores. Then each CPU core calculates its own source and starts the simulation. The second option is to read the partitioned source directly from the file system generated by PetaSrcP. This option is always utilized to handle the huge source input in Petascale level earthquake simulation.

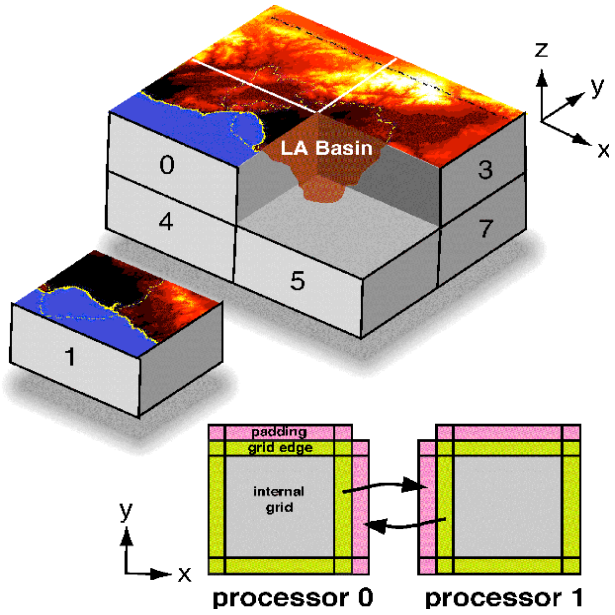


Figure 3.5: AWP main computation running on supercomputers is utilizing tens of thousands of processors. The 3D simulation domain is decomposed into many small 3D grids, which are mapped into those processors. Data communication between processors close to each others occurs during the computation due to the computation characteristics.

The AWP main computation starts after the mesh reader and initial source reader finishes the input processing. This is the core part of the AWP-ODC software, which computes the time serial wave propagation information for the whole 3D domain. Each timestep in the main computation includes two steps: the first step is to compute velocity information based on the stress information, and the second step is to compute new stress based on the updated velocity information. The computation was primarily running on CPU-based supercomputers earlier. Chapters 4 through 6 focus on the heterogeneous solution and present the results.

As the AWP computation lasts for several hours for large-scale earthquake simulation, fault tolerance is required to deal with hardware or system issues. For example, if one computing node died during our computation, then the whole simulation would stop or crash because MPI messaging is blocked. In this case, the simulation will need to be restarted. Without fault tolerance feature, all internal state information of a previous run will be lost. Therefore, checkpointing is fundamental and implemented in AWP, and all the internal state variables are saved periodically to provide a restart capability. Since the mesh and source information will be read again in the restart simulation, we only record the time-related information including veloicty, stress and some intemediate variable information.

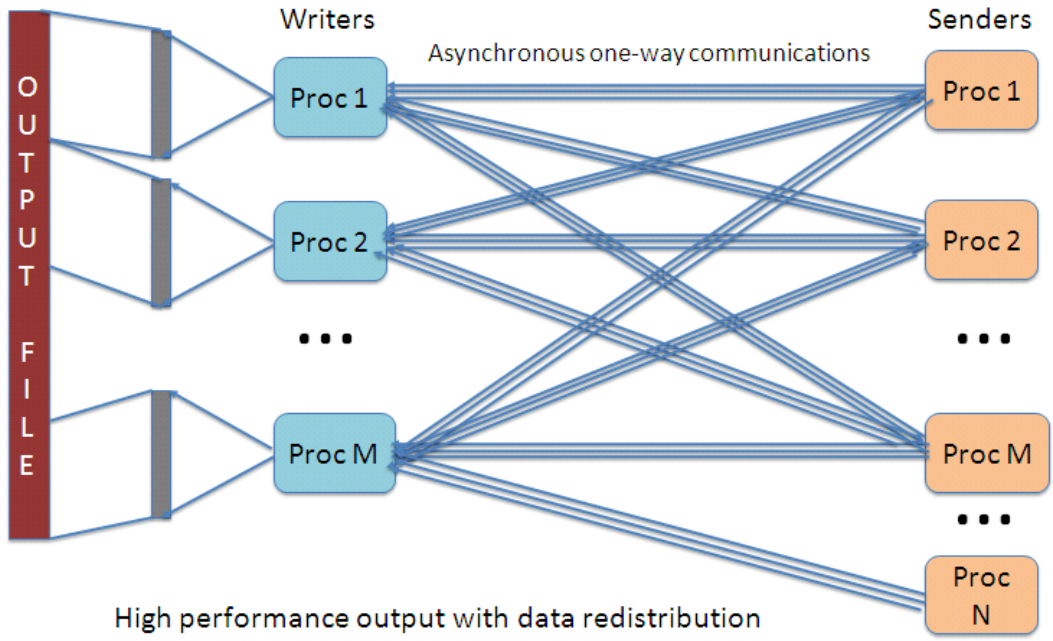


Figure 3.6: Improved AWP high performance output programming model. The concept is very similar as the parallel mesh read. First each core redistributes data into its right output core, which has a big continuous memory chunk. Then write these big chunk data into the file system with high throughput.

After the AWP simulation code finishes the entire computation, the output data must be written into the file system for further scientific study. The output data includes three velocity files for the x, y, and z directions separately. In order to record the time serial information, velocity data must be saved every N timesteps (N is configured before the simulation). Therefore, the output data is also in the hundreds of gigabytes range or even larger. The size of data to be saved at each timestep is equal to the 3D domain size. All outputs generated by the simulation also need to be carefully archived for future research.

Since our simulation code is using hundreds of thousands of CPU cores, the file system cannot afford many cores to write data into a single file simultaneously, because of the limited number of I/O ports shared across the whole system. Hence, we came up with a solution based on the idea of the parallel Mesh Reader, which is the reverse way of the Mesh Read. The improved solution uses a small number of CPU cores to write data into the file system in parallel instead of writing from all CPU cores. Before writing, data is redistributed via MPI communication and packed into big continuous memory data blocks (shown in Figure 3.6). Therefore, there is no data block that overlaps between output CPU cores, which guarantees high throughput on the parallel file system.

After the simulation is completed successsfully, post-processing is required to help validate results, visualize the outputs, and archive the files. Our derived products aVal (Automated Verification) compares new simulation results to the "correct" results from the reference solution by least-squares fit (L2 norm) [54]. Amit Chourasia from our team at SDSC has special visualization tools to generate earthquake wave propagation movies [55]. dPDA and PIPUT are data management script tools developed based on iRODs, which will be discussed more in Section 3.3.

3.3 End-to-End Earthquake Simulation Workflow

The goal is to develop a scientific workflow, that is designed to support SCEC PetaScale simulations on XSEDE and intended to manage jobs and data for the AWP-ODC simulation application. As the complexity and size of the simulation grows, increasing sizes of input or output datasets threatens to outpace the ability to archive and transfer them from site to site in this workflow. As discussed in Section 3.2, SCEC PetaScale wave propagation simulation on the supercomputers typically processes two major input data files with a total size in the multi-terabyte ranges, and generates hundreds of thousands of multi-gigabyte output files in a few hours. Also, based on the scientific workflow design shown in Figure 3.7, tremendous volume of partitioned data files must be transferred across XSEDE. Hence the challenge of improving the performance of this workflow lies in the demand for fast, efficient and reliable data transfer and management.

High performance data transfer and ingestion in grid applications has always been an active area of research. GridFTP [56-57] is a well-known and popular data transfer tool based on Globus that produces high-performance, secure and efficient data transfer technologies optimized for high-bandwidth wide-area networks in Grid environments. Reliable File Transfer (RFT) Service [58], a transfer service developed for grid applications, addresses a wide variety of problems such as dropped connections, machine reboots and temporary network outages automatically via retrying. Grid Datafarm (Gfarm) [59], an architecture designed for petascale data-intensive computing, exploits local I/O in a grid of clusters with hundreds or thousands of nodes and achieves high data transfer rate by parallel I/O read and write operations. Kangaroo [60], a simple data movements system, makes opportunistic use of disks and networks to improve

end-to-end data movement performance in grid environments. However, tools or systems described above mostly focus on data transfer between machines but have no data monitor or verification capability. In addition, some software packages supporting distributed storage environment on XSEDE, such as the leading iRODS software [61], typically utilizes incompatible and unpublished protocols for data transfer.

Based on the characteristics of data types in the scientific workflow and heterogeneous resources on XSEDE, we examine the issues related to performance optimizations of data transfer and ingestion in this scientific workflow to support SCEC PetaSHA simulations. The primary goal is to ensure the correctness of data before running any computation jobs in the workflow and minimize the time-to-solution by reducing data transfer and archiving time.

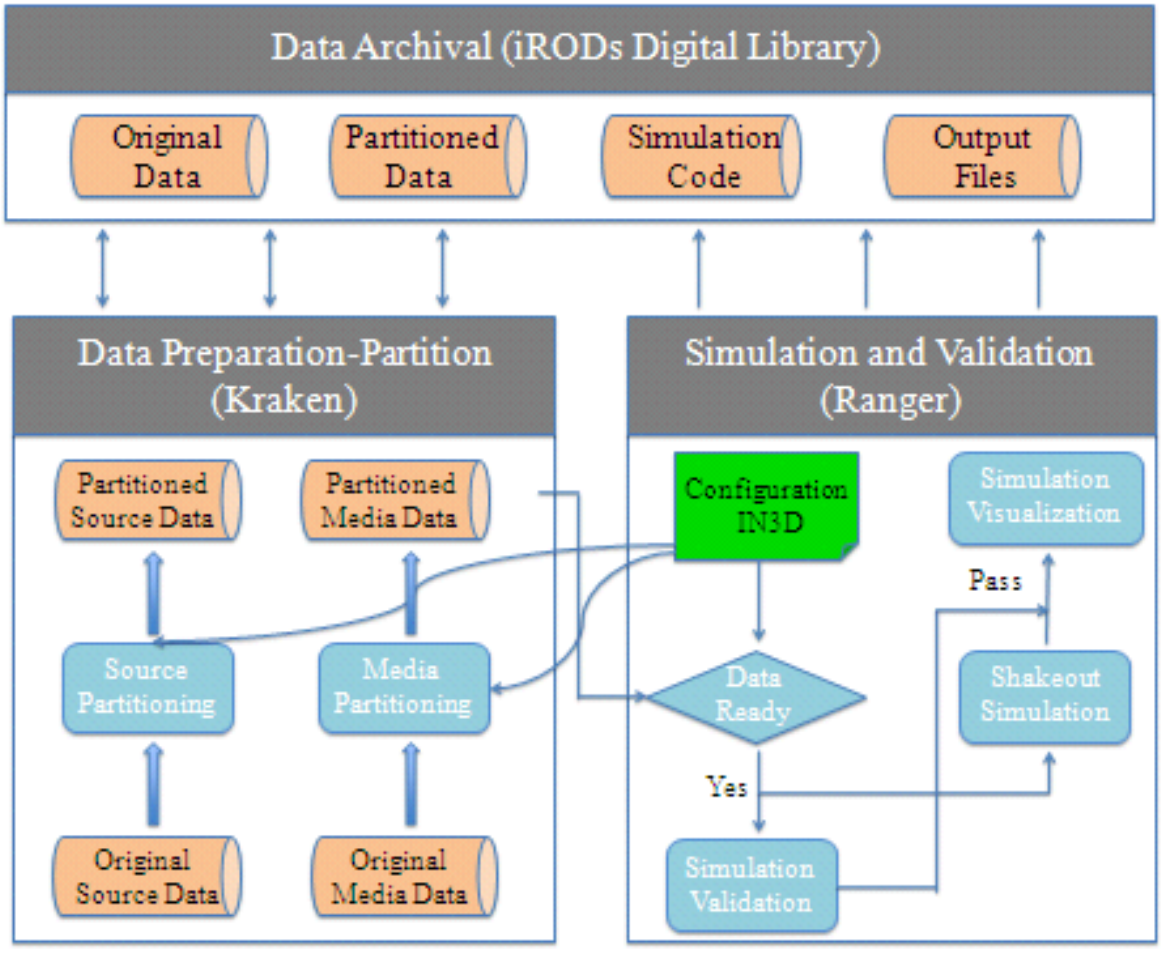


Figure 3.7: System architecture of an end-to-end scientific workflow for a SCEC PetaScale wave propagation simulation. An Shakeout simulation example are presented here: data pre-processing on NICS Kraken, Simulation on TACC Ranger and data archived to iRODs digital library.

Three major components are included in our scientific workflow: data pre-processing, solver simulations and validation, and post-processing data archival. In the Shakeout simulation example as shown in Figure 3.7, two major kinds of data transfer are addressed in the workflow: one is the data transfer between heterogeneous supercomputer clusters on XSEDE, e.g. from NICS Kraken [49] to TACC Ranger, and the other is data archival from supercomputer cluster to iRODS system, e.g. Ranger to iRODS digital library. To investigate these data issues and the feasibility of high performance data transfer and management, we consider integrating advanced data grid tools, remote computation and high-bandwidth networks for solutions. Since data transfer tends to dominate overall simulation performance, the performance evaluation of optimization approaches is focused on transfer or ingestion rate and reliability. All experiment results reported are conducted on the Shakeout Simulation case carried out on XSEDE.

3.3.1 Data transfer between supercomputers

This section presents an enhanced protocol framework that implements data transfer and parallel verification and suggests optimal Globus toolkit [62] parameters for efficient data transfer as part of this framework. This protocol framework has been utilized for high performance data transfer in our scientific workflow. Before describing the framework in some details, we introduce the Globus tool and the commands in use for the framework as well as the functions they performed to help improve the data transfer rate.

The Globus Data Grid Toolkit developed within the Globus project provides a middleware for grid computing environments. Its component Grid Security Infrastructure (GSI) provides public-key-based authentication and authorization services, and resource management services [56]. The GridFTP transfer service uses GSI and supports large amounts of data transfer on Grid. Another feature available is the remote resource access and job management in grid environments. Two important commands of the Globus Toolkit are used in this framework including: globus-url-copy for data transfer and gsissh for remote access management.

For the globus-url-copy command, we have to manually set the TCP buffer size, the number of parallel streams, and the third-party file transfer option with the following syntax:

globus-url-copy –vb –notpt –tcp-bs <buffer> -p <parallel> gsiftp:// <source-machine>/<source-file> gsiftp:// <dest-machine>/<dest-file>

where <buffer> is the TCP buffer size, <parallel> is the number of parallel streams,

<source-machine> is the location for source data files and <dest-machine> is the destination machine for the data files. To avoid high latency and bandwidth problems encountered in networks and maximize the transfer performance, we choose TCP buffer sizes from 1K byte to 64Mbyte and number of parallel streams from 1 to 32 sockets. The optimal parameters supported by Ranger and Kraken are found to be 11Mbyte for the TCP buffer size and 12 sockets for parallel streams. The maximum rate achieved during the data transfer from Kraken to Ranger was 450MB/sec, which is about 33% of the theoretical maximum network bandwidth shared with other TG users (10Gbits/sec).

With the gsissh command, we can access data files on a remote machine, generate the MD5 checksum of each file, and save it to the local machine for data verification with the following syntax:

gsissh <dest-machine> md5sum <filepath> > <local-path>

where <dest-machine> is the remote machine storing source data files, <filepath> is the source data file path on the remote machine, md5sum command is to generate the MD5 checksum and <local-path> is the storage path on the local machine. The MD5 checksum was employed to verify all data files before running any production jobs to ensure the correctness, and the gsissh command can help to generate MD5 checksum from the destination machine to the local machine directly, which costs 50% less time than generating MD5 checksums on the destination machine and transferring them back to local.

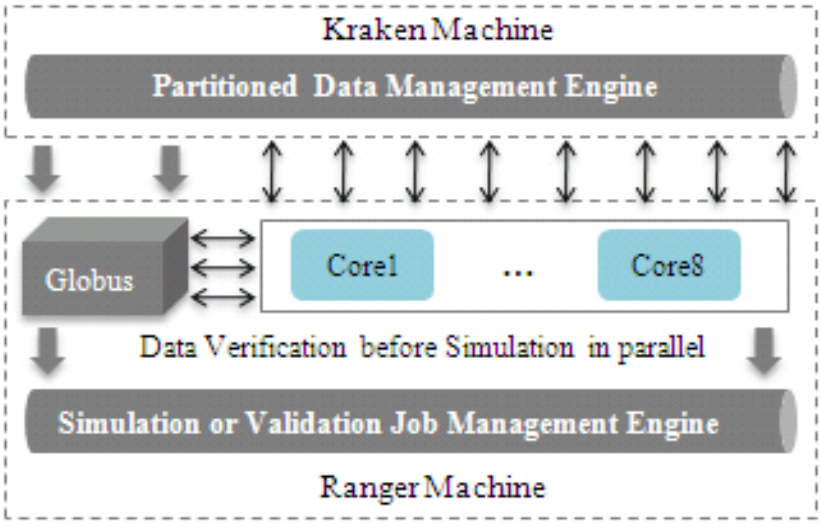


Figure 3.8: Enhanced protocol framework model of high performance data transfer from NICS Kraken to TACC Ranger.

The enhanced framework brings together the Globus toolkit and parallel approaches for high performance data transfer. Figure 3.8 shows a real user case, that utilizes this protocol framework to do high performance data transfer from NICS Kraken to TACC Ranger. Detailed protocols are described as follows:

Step1: TACC Ranger (destination supercomputer machine) sends a request to NICS Kraken (source supercomputer machine) for data transfer with location information. The NICS Kraken checks the data availability and sends a ready signal back to the TACC Ranger, which resembles the handshaking protocol in networking.

Step2: If the ready signal is true, the TACC Ranger enables the Globus tool with 12 threads for source data file transfer. After the network connection is built successfully and the data file start to transfer, TACC Ranger requests another 8 CPU cores using MPI (Message Passing Interface) batch jobs for remote MD5 checksum generation.

Step3: Partitioned Data Management Engine (PDME) in the NICS Kraken allocates data files in equal numbers to each core in TACC Ranger and then remote MD5 checksum generation is executed on the TACC Ranger. This is to ensure each CPU cores on TACC Ranger is generating the MD5 checksum for independent files (e.g. CPU core 0 processes file 0, 8, 16, …, CPU core 1 processes file 1, 9, 17, … and etc).

Step4: When the file data transfer and MD5 checksum generation are finished, these 8 CPU cores on TACC Ranger begin to verify data files in parallel and record information about any incorrect data file.

Step5: Ranger sends this information on incorrect data files to Kraken, and PDME in Kraken locates these data files and resends and verifies them individually until all data files are correct. Then the Simulation or Validation Job Management Engine (SVJME) accesses these transferred data files and starts computing jobs.

In the Shakeout testing case, the number of partitioned data files is 6,400 and the size of each file is around 120 MB, the total size is around 770 GB. The total time spent on data transfer is 3881 seconds and the average transfer rate is 198.40MB/sec. Benefiting from the parallel implementation, the transfer time has been reduced by more than 73%, as the original data transfer and verification took over 4 hours using a single core in sequential order. Additional robustness and reliability are obtained due to the automatic error detection feature implemented in this framework.

3.3.2 Data management and archival

In addition to the high performance data transfer, data management and archival are the other important issues in our scientific workflow. The iRODS system is software middleware that can be tuned to implement any desired data management application, ranging from a Data Grid for sharing data across collaborations, to a digital library for publishing data, to a preservation environment for long-term data retention, to a system for federating real-time sensor data streams [63]. In the Shakeout example (Figure 3.2), we set up an iRODS client on the TACC Ranger, and an iRODS server on SDSC IA64 machine, where the SCEC collection (iRODS digital library) [64] is located on SAM-QFS (Storage Archive Manager- Quick File System) [65] with 16 tape drives and 304 TB disk cache. Normally there are four kinds of data to be managed and archived:

1. Original input data provided by seismologists, over 500GB in size.
2. Partitioned data with total size the same as original data but comprising hundreds or thousands of small data files.
3. Simulation code, configuration files and running scripts for the execution environment.
4. Output data including surface and volume information, in which the size of each volume file is over 10 GB.

Therefore, we set up a strategy as follows to pre-process these data files prior to archival and optimize archival performance to register the data into the iRODS digital library.

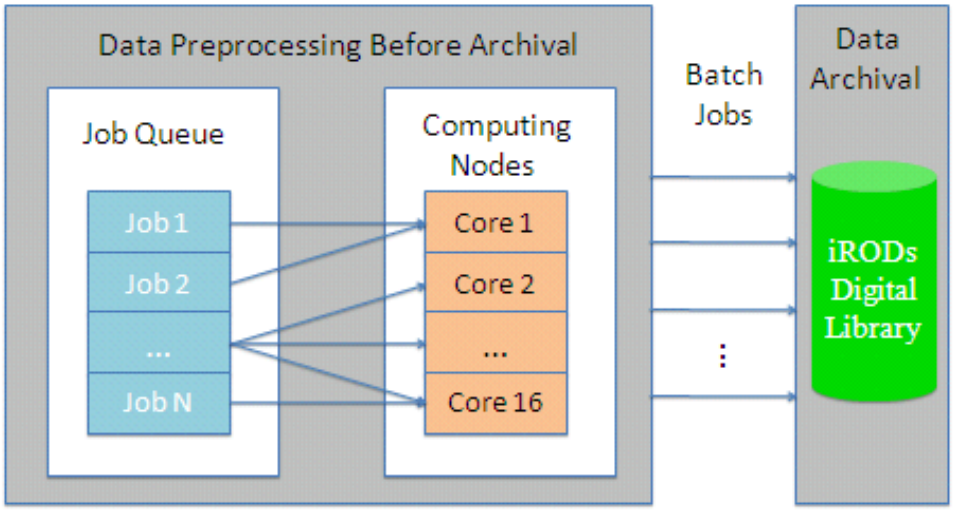


Figure 3.9: Enhanced protocol framework model of high performance data transfer from NICS Kraken to TACC Ranger.

Before data archival, consider three types of jobs in pre-processing: sorting out directories that contains over 1000 files and compressing them individually, selecting single files whose size is larger than 10GB, and combining simulation code, configuration files and running scripts together. Because the tar command may take substantial computation resources, we take advantage of MPI batch jobs to arrange these tasks in parallel to improve the efficiency.

Figure 3.9 shows example data preprocessing for the Shakeout example. In Shakeout simulation, a node including 16 cores on the Ranger machine was requested for data pre-processing. When the simulation was finished, a job queue containing 28 jobs was generated. These jobs were mapped equally to those requested 16 cores and executed in parallel. After each job was finished, the scientific workflow then transferred the pre-processed file to the archival machine via iRODS system automatically.

Transfer of massive data sets to the iRODS digital library was accomplished in usual case by using a single iPUT command. The iPUT command can either ingest a whole directory hierarchy or a set of files. This mechanism is a multithreaded application and adapts the number of threads it uses to the size of the file it transfers. For example, if a file is 32MB or smaller, the iPUT command will utilize a single process and no additional threads will be spawned. If the file has a size between 32MB and 63MB then it will utilize one additional thread. Continuing in this fashion, if the file is of size 512MB or larger then it will use 16 threads, which is the maximum possible number per iPUT command. Our experiments showed that for large files (larger than 3GB) the maximum transfer rate using a single iPUT command with 16 threads is 28MB/sec (evening test with lower system and network load). Thus, our goal was to find ways to improve the ingestion rate and to investigate mechanisms to automate the ingestion and the efficiency of data validation.

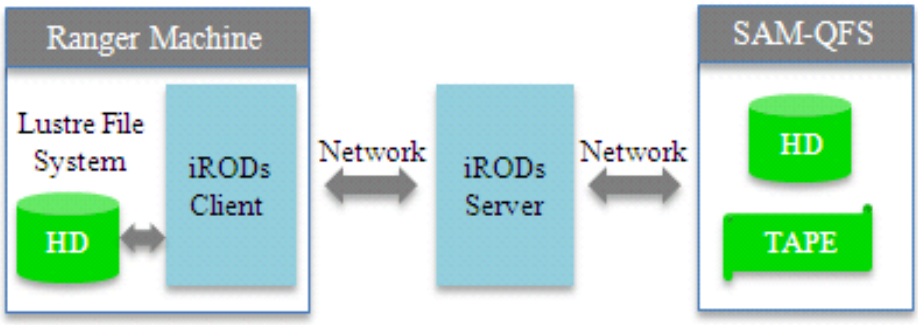


Figure 3.10: Systems involved in the process of data transfer from Ranger to the iRODS digital library.

Figure 3.10 shows the data transfer rate measured as a combined performance result of many systems, where some hardware performances are fixed or depend on file system activities beyond our control. Below is a list of some of these systems involved in the data transfer:

1. Hard disk layout on source machine
2. File system organization and load on source machine
3. CPU utilization
4. Network bandwidth and load
5. iRODS client and server configuration and server load
6. SAM-QFS performance and load

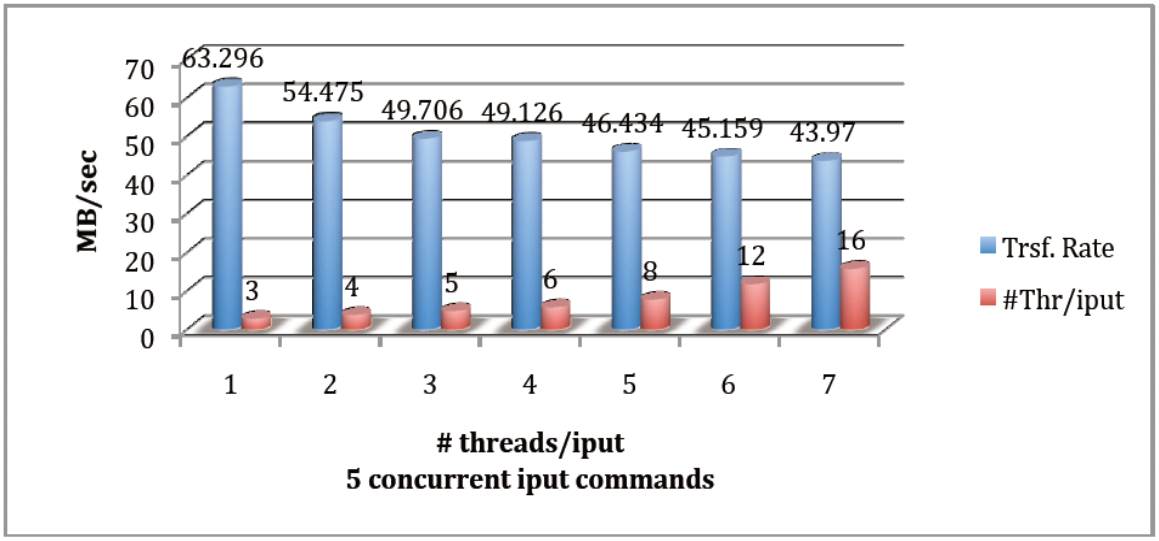
Our approach for improving transfer and ingestion rates focuses on improving CPU utilization and modification of iRODS client and server configurations to improve the network bandwidth utilization. Experiment platforms are the TACC Ranger machine and the SDSC IA64 machine with the iRODS system for the Shakeout example.

There are two alternatives to increase CPU utilization. One is using MPI, and the other is to use the multithreaded capabilities of iPUT and to concurrently invoke many iPUT processes. If one uses MPI then the multithreaded capabilities of iPUT will be lost. Our experiments indicated that 16 threads per iPUT rarely helped concurrently executed iPUT commands, however, 2 threads per iPUT did make a difference that allowed us to improve the rate more than six-fold.

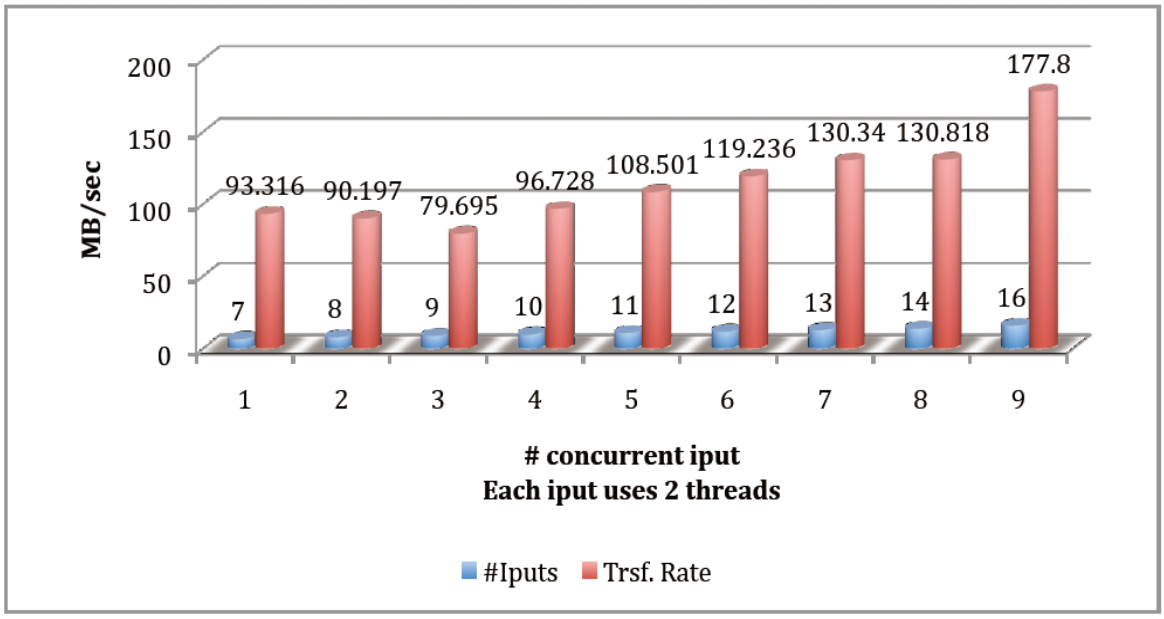
To improve the CPU utilization, we altered the number of parallel iPUT commands and the number of threads each command uses. We were advised to not increase the number of concurrently running iPUT commands beyond 16, since iRODS server would not be able to handle a higher number efficiently, and we experimentally observed degradation in performance beyond 16 iPUT commands.

Figure 3.11(a) presents an ingestion experiment in which we have fixed the number of concurrently executing iPUT commands to five and vary the thread count. The goal is to determine whether higher thread count results in increased performance or not. It shows that the ingestion rate and the thread count are inversely proportional.

In the following experiment we fixed the number of threads per iPUT to two. As shown in Figure 3.11(b), we vary the number of concurrently invoked iPUT commands. With sixteen parallel iPUT and each using two threads, the maximum ingestion rate we achieved is 177.8MB/sec.



(a)



(b)

Figure 3.11: (a) For five concurrent iPUT commands and thread count varied between 3 and 16, best performance is achieved with thread count of three. (b) For two threads per iPUT, maximum rate achieved with 16 iPUT commands running concurrently can be up to 177.8MB/sec

Based on our knowledge, SAM-QFS has only two nodes connecting it to the network with each having 1Gbits/sec bandwidth, hence the combined I/O bandwidth available to SAM-QFS is 2Gbits/sec. Note that 177.8MB/sec is equal to 1422.4Mbits/sec, which means this maximum dataflow rate can be up to 66% of theoretical maximum bandwidth of SAM-QFS.

For iRODS server and client configuration, we explored the effect of the size of the sliding window on the transfer rate. The higher the size of the sliding window TCP uses, the higher the ingestion rate, because the client can stream data up to the size of the window without expecting an ACK from the server. The iRODS server and client are flexible and can vary the size of the sliding window. The TCP window option, defined in RFC 1323 [66] can be used to increase the maximum window size from 65,535 bytes to 1 Gigabyte. Currently the iRODS server and client can use up to a 16MB window size.

On the iRODS server side, the size of the sliding window can be adjusted by modifying the last parameters of microservice in msiSetNumberThreads. If it is set to 16,777,216, then the server will use a 16MB window size. On the iRODS Client's side, the default sliding window size is 1M, and it can also be modified by setting SOCK_WINDOW_SIZE to 16*1024*1024 (16MB).

These two optimizations were combined into our data transfer workflow and experiments demonstrated an average rate of 133MB/sec ingestion rate achieved from TACC Ranger to iRODS digital library, which is nearly five times as the maximum rate achieved per iPUT command (28MB/sec).

3.4 Conclusions

In this chapter, we have presented high performance data transfer and ingestion carried out in a scientific workflow to support SCEC petascale simulations on XSEDE. The scientific workflow for the Shakeout simulation case is an outstanding example that shows the efficiency of our optimizations provided for SCEC Petascale simulations. This chapter's contributions are as follows.

We have suggested best parameters for Globus toolkits to maximize data transfer performance. An outstanding feature is the protocol framework for data transfer between supercomputer clusters on XSEDE. We have also utilized advanced MPI batch jobs to generate MD5 checksums and verify data files in parallel. The total data transfer time has been reduced by more than 73% based on these parallel implementations.

For data archival from supercomputer clusters to iRODS digital library, we developed a new strategy for data management and pre-processing. A job queue is created to sort out directories containing hundreds of thousands of files, pick up large data files and tar small files

together. MPI batch jobs are utilized to help run these jobs in parallel to shorten the running time and reduce the load on supercomputer clusters. When archiving data from Ranger to the iRODS system, we improved CPU utilization and modified the iRODS server and client configuration. We achieved an average transfer rate of 133MB/sec, which is nearly five times faster than the conventional iRODS method.

Acknowledgements

Section 3.1 and 3.3, in part, are a reprint of the material as it appears in the 2010 third international joint conference on Computational Science and Optimization with title "Workflow-Based High Performance Data Transfer and Ingestion to Support Petascale Simulations" by Jun Zhou, Yifeng Cui, Sashka Davis, Clark C. Guest, Philip Maechling. Section 3.2, is based on the material as it partly appears in proceedings of the 2010 ACM/IEEE conference on Supercomputing with title "Scalable Earthquake Simulation on Petascale Supercomputers" by Yifeng Cui, Kim B. Olsen, Thomas H. Jordan, Kwangyoon Lee, Jun Zhou, Patrick Small, Daniel Roten, Geoffrey Ely, Dhabaleswar K. Panda, Amit Chourasia, John Levesque, Steven M. Day and Philip Maechling. The dissertation author was one of the primary investigators and author of these two papers.

Chapter 4

Single GPU Optimization

This chapter presents our hands-on performance tuning experience and describes optimization approaches for higher computational efficiency of 13-point asymmetric 3D stencil based Finite Difference code used in AWP-ODC, with more focus on aggressive performance for the second generation NVIDIA GPU "Fermi" chipset. We completely rewrote AWP-ODC in C and CUDA in order to take advantage of the powerful GPU computing capabilities. We present performance comparisons between our fully optimized AWP-ODC Fortran MPI code running on different multi-core CPU systems and the new CUDA code running on different GPU chipsets. Benchmarks on NVIDIA Tesla M2090 demonstrated 10 times speedup versus the original fully optimized AWP-ODC FORTRAN MPI code running on a single Intel Nehalem 2.4 GHz CPU socket (4 cores/CPU), and 15 times speedup versus the same MPI code running on a single AMD Istanbul 2.6 GHz CPU socket (6 cores/CPU). Sustained single-GPU performance of 143.8 GFLOPS in single precision is benchmarked for the testing case of 128x128x960 mesh size on the NVIDIA Tesla M2090 GPU.

NVIDA GPU architectures for scientific computation have gone through three generations including Tesla 10x series, Fermi, and Kepler. All optimization approaches described in this chapter are mainly based on the Fermi, since Kepler was not available during the time we worked on single GPU code porting. However, most of these approaches work very well on the Kepler architecture. Benefit from the faster read-only memory and more flexible shared memory, further optimization research work could make our computing code running faster on this new chip, which will be described in the future work section. For the same reason, some optimization approaches cannot be run on the Tesla 10x series because of the architecture differences, e.g. there is no L1 cache on Tesla 10x GPUs.

In this Chapter, we first will describe the analysis of the 13-point asymmetric 3D stencil kernels. Then brief summary of our CPU optimization and results will be presented. After that, we will discuss our optimization approaches in detail using both the algorithm and experiments. Finally, performance comparisons are shown to demonstrate excellent optimization results.

4.1 AWP-ODC Kernel Analysis

This section describes the formulation of the AWP-ODC numerical model and analysis of the computation kernels. AWP-ODC solves a 3D velocity-stress wave equation using an explicit method with a staggered-grid finite difference method, fourth-order accurate in space and second-order accurate in time. The coupled system of partial differential equations includes the particle velocity vector v and the symmetric stress tensor σ [5]. Let:

$$v = (v_x, v_y, v_z) \tag{4.1}$$

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \tag{4.2}$$

Then the governing elastodynamic equations are [1]:

$$\partial_t v = \frac{1}{\rho} \nabla \cdot \sigma \tag{4.3}$$

$$\partial_t \sigma = \lambda (\nabla \cdot v) \mathrm{I} + \mu (\nabla v + \nabla v^T) \tag{4.4}$$

λ and μ are the Lame coefficients and ρ is the constant density. Simplifying formulae (4.3) and (4.4) lead to three scalar-valued equations for velocity vector components and six scalar-valued equations for the stress tensor components, which are listed below:

$$\frac{\partial v_x}{\partial t} = \frac{1}{\rho} \left(\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \frac{\partial \sigma_{xz}}{\partial z} \right) \tag{4.5}$$

$$\frac{\partial v_y}{\partial t} = \frac{1}{\rho} \left(\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{yz}}{\partial z} \right) \tag{4.6}$$

$$\frac{\partial v_z}{\partial t} = \frac{1}{\rho} \left(\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} \right) \tag{4.7}$$

$$\frac{\partial \sigma_{xx}}{\partial t} = (\lambda + 2\mu) \frac{\partial v_x}{\partial x} + \lambda \left(\frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \right) \tag{4.8}$$

$$\frac{\partial \sigma_{yy}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_y}{\partial y} + \lambda(\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z}) \tag{4.9}$$

$$\frac{\partial \sigma_{zz}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_z}{\partial z} + \lambda(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}) \tag{4.10}$$

$$\frac{\partial \sigma_{xy}}{\partial t} = \mu(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}) \tag{4.11}$$

$$\frac{\partial \sigma_{xz}}{\partial t} = \mu(\frac{\partial v_x}{\partial x} + \frac{\partial v_z}{\partial z}) \tag{4.12}$$

$$\frac{\partial \sigma_{yz}}{\partial t} = \mu(\frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}) \tag{4.13}$$

AWP-ODC is a memory-intensive application using twenty-one 3D variable arrays in the main computation loop. In AWP-ODC, two computation kernels for velocity and stress are carried out in sequence for wave propagation simulation based on numerical approximation of the partial differential equations. At each time step in the main loop, for each mesh point in the domain, the velocity computation kernel updates three velocity components (in X, Y, and Z directions) by using six stress components (on XX, YY, ZZ, XY, XZ, and YZ faces), and then the stress computation kernel employs these updated velocity components to update six stress components. We have twenty-one 3D arrays to be maintained in the memory to process the wave propagation, including velocity, stress and coefficients. In addition to the three velocity vector components and six symmetric stress tensor components, 6 temporary variables (r1, r2, r3, r4, r5, r6) and 6 constant coefficients (λ, μ, ρ, quality factor for S wave Qs and P wave Qp, boundary condition variable Cerjan Cj [67]) are utilized in the numerical modeling. The size of each 3D array is the same as the 3D simulation domain, hence effective memory fetching is the key to achieving high computation performance. Figure 4.1 is the detailed pseudo-code of the computation kernels in the main loop based on the numerical approximation of the formulae 4.5 ~ 4.13:

Main Loop:

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

Compute velocities (vx, vy, vz) using stress (xx, yy, zz, xy, yz, xz)

Update values of velocities (vx, vy) along the surface

Update values of velocities (vz) based on (vx, vy) along the surface

Compute stress (xx, yy, zz, xy, xz, yz) based on velocities (vx, vy, vz)

Update values of stress (zz, xz, yz) along the surface

END DO

Velocity Computation Kernel (only vx computation is shown here, similar for vy & vz)

```
vx(i, j, k) += d1(i,j,k)*( c1*(xx(i, j, k) – xx(i-1, j, k)) + c2*(xx(i+1, j, k) – xx(i-2, j, k))
                           c1*(xy(i, j, k) – xy(i, j-1, k)) + c2*(xy(i, j+1, k) – xy(i, j-2, k))
                           c1*(xz(i, j, k) – xz(i, j, k-1)) + c2*(xz(i, j, k+1) – xz(i, j, k-2)) )
```

Stress Computation Kernel 1 (only xx computation is shown here, similar for yy & zz)

```
vxx = c1*( vx(i+1, j, k) – vx(i, j, k) ) + c2*( vx(i+2, j, k) – vx(i-1, j, k) )
vyy = c1*( vy(i, j, k) – vy(i, j-1, k) ) + c2*( vy(i, j+1, k) – vy(i, j-1, k) )
vzz = c1*( vz(i, j, k) – vz(i, j, k-1) ) + c2*( vz(i, j, k+1) – vz(i, j, k-2) )
tmp = (xl + d_DT*qpa)*(vxx + vyy + vzz)
a1  = d_DT*qpa*(vxx + vyy + vzz)
xx(i, j, k) = (xx(i, j, k) + tmp – xm*(vyy + vzz) + vx1*r1(i, j, k))*dcrj(i, j, k)
r1(i, j, k) = x2(i, j, k)*r1(i, j, k) - h(i, j, k)*(vyy + vzz) + a1
```

Stress Computation Kernel 2 (only xy computation is shown here, similar for xy & yz)

```
vxy = c1*( vx(i, j+1, k) – vx(i, j, k) ) + c2*( vx(i, j+2, k) – vx(i, j-1, k) )
vyx = c1*( vy(i, j, k) – vy(i-1, j, k) ) + c2*( vy(i+1, j, k) – vy(i-2, j, k) )
xy(i, j, k) = xy(i, j, k) + xmu1(i, j, k)*(vxy+vyx) + x1*r4(i, j, k)
```

Figure 4.1: AWP-ODC pseudo code for computation kernels: vx stands for velocity in x direction. xy stands for stress xy component. c1, c2 and x1 are scalar constants. r1 and r4 are intermediate variables and also updated in stress calculation. d1 is the constant density ρ, while xm, xl, xmu1 qpa, h, h1and x2 are 3D static lame coefficient variables derived from λ and μ.

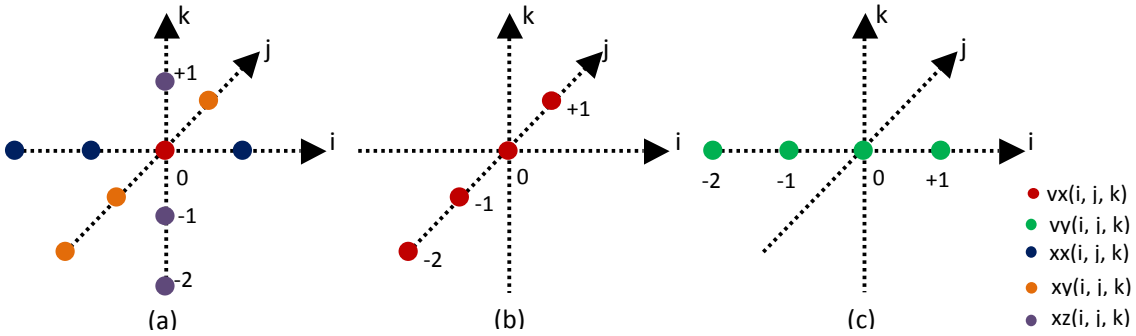


Figure 4.2: Analysis of AWP-ODC Computation Kernels based on the pseudo-code in Figure 2: (a) is the velocity computation kernel for vx, 13 point asymmetric stencil computation involving 3 stress components. (b) and (c) are for the stress component xy computation, two asymmetric stencils in x and y directions respectively. Asymmetric stencil computation is the same as the regular stencil computation for single CPU/GPU programming, but the asymmetric property is good for MPI optimization for multi-CPUs or GPUs programming.

The 13-point asymmetric stencil computation for vx in the AWP-ODC main loop is shown in Figure 4.2a, where reads occur 12 times in three different 3D arrays and writes occur only once in one 3D array. Stress component xy calculation includes two 1D asymmetric stencils (Figure 4.2b and 4.2c), with only 4 reads from a single 3D array and also one writes in one 3D array. Approximately 136 reads, 15 writes and 307 FLOPs calculations in total are involved for each point of the 3D domain in a single iteration. Moreover, data access for these reads and writes occurs in the twenty-one 3D arrays described before, and each computation kernel involves more than three 3D variable arrays. Table 4.1 summarizes the analysis of the three kernels in Figure 4.2, showing that AWP-ODC is a memory-bound application because of the low FLOPS to bytes ratio (the average operation intensity is around 0.5), which means the application has poor temporal data locality and the performance is dominated by the memory system or arithmetic throughput [68]. Again it shows that improving the data locality has been the key to achieve the high computing performance for AWP-ODC kernels.

Table 4.1: Analysis of Computation Kernels in AWP-ODC Main Loop

Kernels	Reads	Writes	FLOPs	FLOPs/Bytes
Velocity Computation Kernel	51	3	86	0.398
Stress Computation Kernel	85	12	221	0.569
Total	136	15	307	0.508

4.2 AWP-ODC CPU Implementation/Optimization

The AWP-ODC CPU implementation and optimization works was done in 2010 by a team led by Prof. Yifeng Cui at SDSC including myself. Details are published in our SC'10 paper [5]. Here I will present some summary for further comparison between CPU and GPU optimization. The AWP-ODC CPU implementation/optimization includes three main parts: 1. reducing the expensive operations; 2. cache blocking for better data locality; and 3. loop unrolling to improve cache utilization. On OLCF Jaguar CPU-based supercomputer in 2010, the performance gain was around 40% at full system scale, with 31% from arithmetic optimization, 7% from cache blocking and 2% from loop unrolling.

In the numerical formulae, the coefficients mentioned in Section 4.1 are constant in the entire simulation. However, the original numerical model requires reciprocal form of these coefficients, such as the lam and mu (coefficients for S wave and P wave), which are computed in the form 1/lam(i, j, k) and 1/mu(i, j, k) instead of lam(i, j, k) and mu(i, j, k). Therefore, pre-computed reciprocal values for these coefficients are stored and used in the main computation loop to avoid these expensive division operations [69]. Another reduced expensive operation is the "mod" function. We eliminated the mod function from the innermost computing loop by replacing it with itx = 3 - itx, where the value of itx alternates between 1 and 2. Removal of the mod enabled the compiler to vectorize the arithmetic in a compute-intensive loop.

The critical subroutines share the same three nested loop structures. Each node executes the loop over its processor's local mesh, which achieves good memory access behavior. However, the cache utilization rate is very low, primarily due to the requirement of assessing values in multiple 3D arrays with varying second or third indices. When one of these values is accessed, the whole cache line containing the value is fetched into the L1 cache. Since the number of variables in the inner loops is large, a cache line is usually evicted from the L1 cache right after being referenced.

To improve cache utilization, we need to access as many values per cache lines loaded as possible. The cache blocking technique provides better cache utilization. A 3D difference algorithm can be extremely limited in memory bandwidth, and the number of variables that must be fetched from memory is relatively high given the computation performed. If the required operands are in cache, the effectiveness of cache reuse in the difference code is low since the

amount of data during the computation of an entire plane will exceed the size of the L1 and L2 caches. Consider the AWP-ODC CPU code is developed in Fortran code, so the memory access pattern for each 3D variable array mu(i, j, k) is fast in i direction and slow in k direction. Here is an example showing how we implement the cache blocking for 3D 13-point stencil computation:

```
Do k = nzb, nze
  Do j = nyb, nye
    Do i = nxb, nxe
      xm = 8.0/( mu(i, j, k) + mu(i+1, j, k)
           + mu(i, j-1, k) + mu(i+1, j-1, k)
           +mu(i, j, k-1) + mu(i+1, j, k-1)
           +mu(i, j-1, k-1)+mu(i+1, j-1, k-1) )
    End Do
  End Do
End Do
```

(a)

```
Do kk = nzb, nze, kblock
  Do jj = nyb, nye, jblock
    Do k = kk, min(kk+kblock-1, nze)
    Do j = jj, min(jj+jblock-1, nye)
    Do i = nxb, nxe
      xm = 8.0/( mu(i, j, k) + mu(i+1, j, k)
           + mu(i, j-1, k) + mu(i+1, j-1, k)
           +mu(i, j, k-1) + mu(i+1, j, k-1)
           +mu(i, j-1, k-1)+mu(i+1, j-1, k-1) )
    End Do
    End Do
    End Do
  End Do
End Do
```

(b)

Figure 4.3: Detailed example shows the cache blocking optimization for AWP-ODC CPU Fortran code. The left (a) is before cache blocking and the right (b) is after cache blocking, where the block size is decided by kblock and jblock.

For any reasonably sized grid in Figure 4.3a, the lines containing the variables from the j-1 and k-1 planes will not be located in cache when the difference equation is performed on the next plane. If the grid is subdivided into smaller sub-grids, the operands from the j-1 and k-1 planes may still be in cache as the computation progresses. As shown in Figure 4.3b, the values of kblock and jblock are chosen to guarantee that the operands on subsequent planes are still in cache while those planes are computed, so that each grid point will be accessed eight times. If the cache blocking is perfect, then a variable will only be fetched from memory once and the other seven fetches will be from the L1 or L2 cache. The values of kblock and jblock are dependent upon the number of operands accessed within the loop and the cache/cache line size on the

architecture. For our M8 simulation on Jaguar, the optimal solution was found to be 16/8 for the j/k direction.

Loop unrolling techniques are also utilized to improve cache utilization and data locality, even though some compilers automatically incorporate this form of optimization. Due to the limited number of registers on a processor, unrolling too much could deteriorate loop performance. In the AWP-ODC CPU Fortran code, unrolling by 2 iterations provides the best performance for the computing-intensive subroutines of stress xy and xz.

4.3 NVIDIA GPU Computing Architecture

In the high performance computing area, the NVIDIA GPU computing architecture has gone through three generations including the 10-series, Fermi and Kepler (More details in Section 2.1). All these three architectures have the exact same computing and memory access pattern. The principal difference is the computing capability and feature support. For example, the Kepler architecture has more CUDA cores in each streaming multiprocessor (192 CUDA cores per SMX vs 32 cores on Fermi vs 16 cores on 10-series), and also supports new features such as "Dynamic Parallelism", which allows threads running on the GPU to create another child thread (not supported by Fermi and 10-series). Other than that, the programming model is substantially the same for these three NVIDIA GPU architectures.

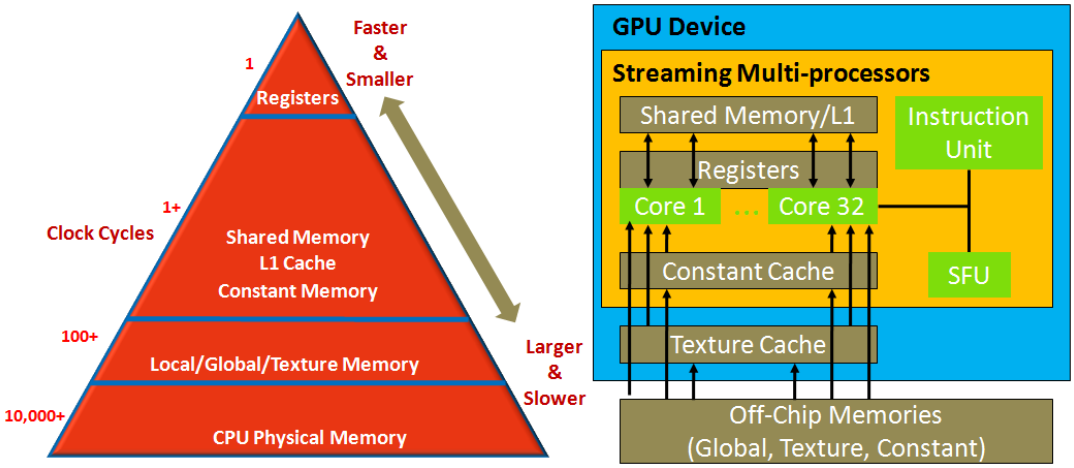


Figure 4.4: NVIDIA GPU memory architecture hierarchy: registers are fastest but are limited in number per CUDA thread (64 on Fermi), while the large CPU physical memory is the slowest due to the PCI Express connection. Minimizing data access to global/CPU memory is one of the key factors to improve computing performance [70-71].

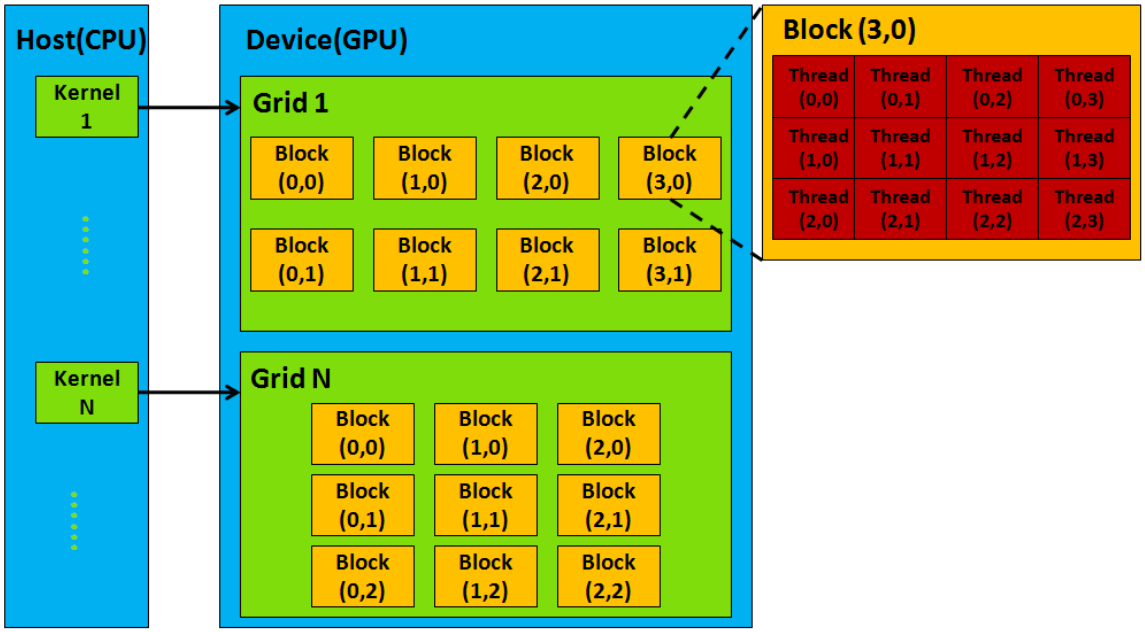


Figure 4.5: NVIDIA GPU threading model: Single Program Multiple Data. Each kernel is decomposed into multiple blocks and each block is running on the physical streaming multi-processor (SM). Each block contains hundreds or thousands of threads, and the basic unit for each block is a warp, which is 16 for 10-series and 32 for Fermi and Kepler. Threads in the same block are running on the SM in the format of warps [70-71].

In SIMD, we need to avoid branching and hanging/waiting data fetch, but keep all CUDA cores busy on computation. Based on the memory hierarchy in Figure 4.4, if all threads in the same warp can fetch data from the fast on-chip memory in a single cycle and execute the computation for a long time, ideally we can achieve performance close to the theoretical value. Therefore, improving data locality is one of the most important optimizations for GPU computing.

Figure 4.5 shows the NVIDIA GPU single program multiple data (SPMD) threading model. Generally, kernels have to run based on the submission order if there is dependence between them, and only a single kernel takes up all the computing resource. Other kernels cannot run until it finishes its work. In the latest Kepler architecture, each SMX contains multiple schedulers, which permits kernels from independent streams or CPU threads to run concurrently. However, AWP-ODC only contains two dependent homogenous computation kernels, so these new Kepler features cannot provide much help from the performance perspective other than more CUDA cores. Hence increasing the computing throughput is another key factor to maximize the computation performance on all GPU platforms for AWP-ODC.

4.4 Single GPU Implementation and Optimization

This section describes the AWP-ODC GPU based code implementation and performance tuning strategies on a single NVIDIA GPU M2090, the Fermi Architecture with 2.0 computing capability. The NVIDIA Fermi Architecture is very mature and popular in the current HPC market (2012~2013). All optimization methods can be directly applied to the latest Kepler architecture as well. The NVIDIA Tesla M2090 was released in May 2011, which has 512 CUDA parallel processing cores, and delivers 1,331 GFLOPS in single-precision (SP) performance, with 6 GB GDDR5 memory size and 177 GB/sec memory bandwidth (ECC off) [18]. For this Fermi Chipset, each streaming multiprocessor (SM) includes 32 CUDA cores, along with 16 load/store units for memory operations to improve I/O performance, 4 special-function units (SFU) to handle complex math operations and 64KB local SRAM split between hardware-managed L1 cache and software-managed shared memory. The local SRAM can be split according to two different modes: 16KB/48KB or 48KB/16KB based on the users' requirement. One of the partitions, "shared memory", is the fast memory that can be accessed by all 32 cores in the same SM. Each Fermi Chipset also has a 768KB L2 cache shared by all SMs. In the L2 cache subsystem, the atomic instructions (a set of read-modify-write memory operations) have been improved to 5 to 20 times faster than the NVIDIA first generation GPU chipset [70].

The GPU codes discussed in this section are developed in C and CUDA languages, and compiled by the CUDA compiler nvcc 4.0, with options: "-O4 –Xptxas –dlcm=ca –maxrregcount=63 –use_fast_math –arch=sm_20". Two extra layers are added to each direction (called ghost cells) in the kernel stencil computation, which means the actual size for the 3D grid in memory is (NX+4, NY+4, NZ+4) instead of (NX, NY, NZ). The value in the ghost cell regions are based on the "equal to the nearest value" rule, which means the mesh point values in (-1:0, NY, NZ) equals to the ones in (1, NY, NZ).

When the AWP-ODC GPU program starts, the CPU initializes and allocates all twenty-one 3D variable arrays from input files or parameters, then copies all these 3D arrays into the GPU device memory via the PCI 2 Express bus and executes the velocity and stress kernels as shown in Figure 4.1. After the main loop computation is finished, result data are transferred back from GPU to CPU main memory for output. Our optimization mechanisms will be focus on five steps for performance tuning.

4.4.1 Read-Only Memory Cache

Constant and texture memory in the GPU device have their own cache (Figure 4.4) and both are only for read-only data. Constant memory in the NVIDIA GPU is hardware optimized for broadcasting, especially when all threads from a same warp read the same constant memory location. We can consider some optimization using the constant memory for its temporal locality. However, the performance will drop dramatically if threads in the same warp access different constant memory allocations because accesses are serialized. In general, input parameters for each kernel will be put into the constant memory as well as constant values used in the kernel, i.e. "if (a > 1.0)" where the 1.0 is put into the constant memory by the compiler. To make full use of this temporal locality provided by the GPU constant memory, scalars and small coefficient constants with fixed access pattern are always recommended for storage in constant memory.

Texture memory is optimized for spatial locality for both 2D and 3D textures, and the addressing calculations can be performed by hardware outside of the kernel. Because of the 2D and 3D locality, cache line fills can pull 2D and 3D blocks from memory instead of rows, perfect for stencil computation and filtering. Unlike the constant memory, all threads from the same warp can access different texture memory locations without performance penalty. Moreover, 8 bit and 16 bit data converting to floating point numbers between 0.0 and 1.0 is done for free (good for interpolation), and boundary conditions can be dealt with by the texture piping for free as well. Therefore, texture memory is great for large blocks of read-only data or arrays, where each data will be accessed or shared by multiple threads.

The physical location for both constant and texture memory is the same as the global memory, so it will not save any memory space if we put some data into the constant/texture memory instead of the global memory. In addition, the coalesced data access pattern also must be optimized for texture memory, otherwise high cache miss rate will kill the computation performance due to the memory-bound application property.

As discussed in Section 4.1, six 3D variable arrays are constant coefficients and the access pattern for each variable is 13-point stencil, so we put all of them into the texture memory to take advantage of the texture cache for 3D spatial locality. All scalar constants (such as c1 and c2) in Figure 4.1 are put into constant memory for temporal locality. This also helps us to save register usage. All other 15 arrays have to be stored in global memory because their values are being updated during iterations.

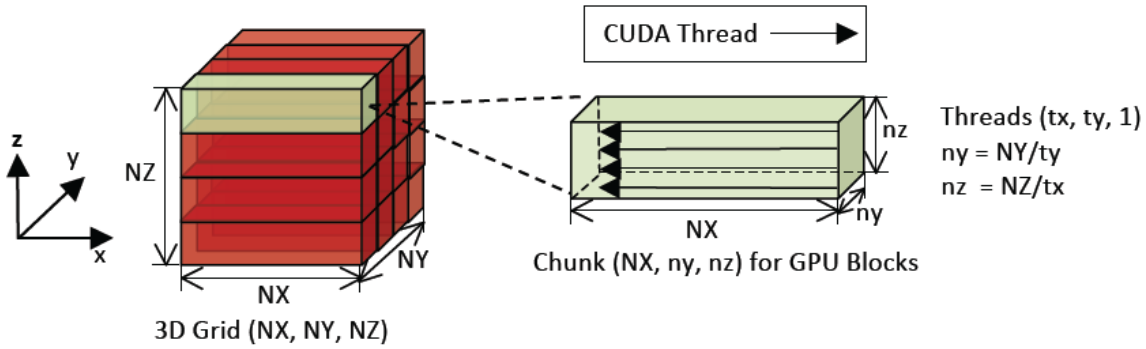


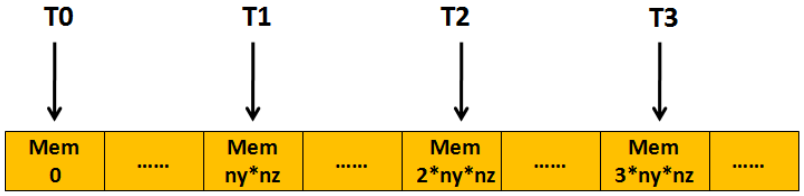
Figure 4.6: Domain decomposition for GPU kernels: the 3D Grid (NX, NY, NZ) is decomposed only in the y and z directions. Suppose each block has (tx, ty, 1) threads, then the kernel function has (NZ/tx, NY/ty, 1) blocks and the chunk size for each block will be (NX, ny, nz).

4.4.2 Domain Decomposition

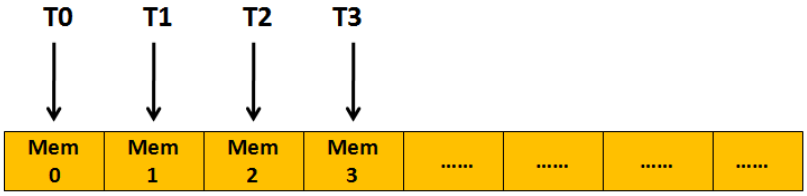
CUDA is an extension language of C/C++, so memory storage for 3D arrays will be fast in the z direction and slow in the x direction. To obtain a better cache hit rate and allow all threads in the same warp to access data along the fast z axis instead of the slow x axis, we decompose the 3D Grid only in y and z directions. Each thread calculates the entire NX for a given 2D (y, z) location as shown in Figure 4.6.

Table 4.2: Performance comparison between two implementations with different decomposition geometry derived from a baseline code, which is based on direct global memory access without any optimization. The benchmark runs for 800 timesteps for time-to-solution measurement. The thread block for GPU Kernels is (64, 8, 1), so the value B_x = 64 and B_y=8 in the table.

3D Grid Size (NX x NY x NZ)	**CUDA CODE for (x, y) 2D Decomposition** 1. x = blockldx.x*B_x + threadldx.x + 2; 2. y = blockldy.y*B_y + threadldy.y + 2; 3. for (z = NZ+1; z>=2; --z) 4. Velocity and Stress Computation	**CUDA CODE for (y, z) 2D Decomposition** 1. z = blockldx.x*B_x + threadldx.x + 2; 2. y = blockldy.y*B_y + threadldy.y + 2; 3. for (x = NX+1; x>=2; --x) 4. Velocity and Stress Computation
128x128x128	0.242 sec/timestep	0.008 sec/timestep
128x128x256	0.490 sec/timestep	0.020 sec/timestep
128x256x256	0.977 sec/timestep	0.040 sec/timestep
256x256x256	1.956 sec/timestep	0.086 sec/timestep



(a) Memory Access Pattern for (x, y) 2D Decomposition



(b) Memory Access Pattern for (y, z) 2D Decomposition

Figure 4.7: Memory access pattern for different 2D Decompositions: (a) is for (x, y) and (b) is for (y, z), where both pseudo codes can be found in Table4.2 column 2 and 3 respectively. "T0" and "T1" means thread 0 and thread 1, and all T0 - T3 belong to the same warp.

The basic unit for GPU computing is the warp and the warp size is 32 in Fermi, which means all 32 threads in the same warp will execute the same instructions. If any thread in the warp needs to wait for its data, then all other threads will be hanging there to wait for the data fetch. Table 2 is the performance comparison between 2D decomposition in the (x, y) direction and the (y, z) direction. The 2D decomposition in (x, y) means threads in the x direction correspond to the 3D Grid x direction, and threads in the y direction correspond to the 3D Grid y direction. While the 2D decomposition in (y, z) means threads in the x direction correspond to the 3D Grid z direction, and threads in the y direction correspond to the 3D Grid y direction. Figure 4.7 shows the memory access pattern for these two decompositions. The thread block is also a three dimensional topology and fast in the x direction, but the memory storage is fast in z direction. Therefore, data requested by the 32 threads in the same warp cannot be continuous and the physical locations are far from each other if decomposed in (x, y) as shown in Figure 4.7a. It will take 32 data fetches from global memory or L3 cache. But if decomposed in the (y, z) directions, data will be close to each other (shown in Figure 4.7b), and single cache line fetch is enough to acquire all data requested by the warp. This has far less memory access latency compared to the (x, y) decomposition. Data in Table 4.2 shows that the code performance for decomposition in the (y, z) direction is 25 times faster than in the (x, y) direction. For best configuration, the number of threads in the x direction is considered to be a multiple of 32 for better performance.

4.4.3 Memory Padding

In GPU computing, memory alignment accessing is also very important for performance, which can make best effective use of the modern DRAM architecture for memory load/store. In general, an address in memory is 2^n bytes aligned, such as 64 bytes with the least significant 6 address bits equal to zero. Therefore, the ideal memory access should have each memory load starting from an aligned address with its size equals to the aligned region.

As discussed earlier, each 3D domain includes two extra ghost cells in each direction around it, which means mesh points outside of the 3D domain are needed in order to compute mesh points in the boundary region. Here we use the mesh point (0, 0, 0) as an example: if this mesh point is at an aligned memory address, then the mesh points (0, 0, -1) and (0, 0, -2) will be outside of the aligned region. Therefore, we pad some additional data to the "-z" direction to make sure these data in the ghost cell region are also within the aligned region.

Moreover, the domain size may not be a power of 2 in the z direction, depending on the earthquake region and simulation model. To guarantee data access is also within the aligned region when the computation starts at the next row, we have to pad some additional data at the end of each row. This padding data is meaningless and will not be used during the computation, but it helps to make sure most memory access starts from an aligned address.

CUDA provides some functions such as "cudaMallocPitch" and "cudaPitchedPtr" to help ensure aligned memory access [71]. However, as shown in Figure 4.8a, instead of using these library functions, we manually pad zeros onto the boundaries in the z axis of the 3D grid to align memory for the inner region. The padding size plus NZ should also be a multiple of 32, and the two layers of ghost cells are included in the padding.

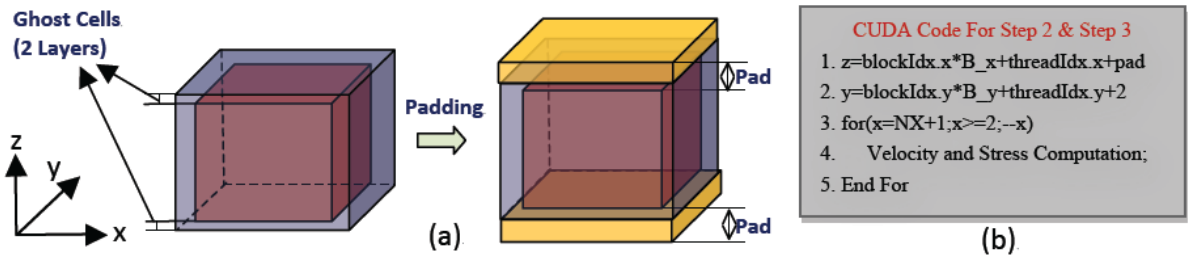
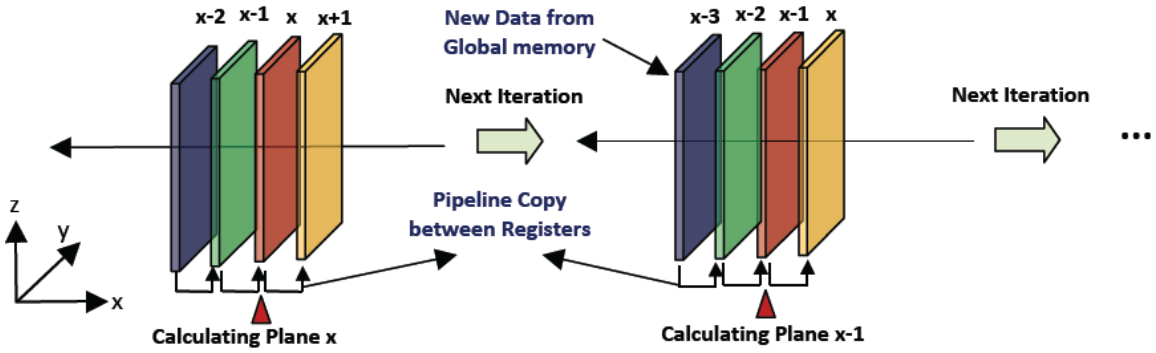


Figure 4.8: (a) 3D array padding: the red cube represents the original 3D grid, the blue cube represents the ghost cells with 2 extra planes in each direction. The yellow part is the padding arrays, including two layers of ghost cells along the z axis. (b) is CUDA code for steps 2& 3, fast z for better memory load efficiency after padding.

4.4.4 Register Optimization

Private registers are the fastest on-chip GPU memory and take only one clock cycle for data access or calculation. Neighboring data in y and z directions are close to each other in aligned memory, therefore these data can be cached or pre-fetched into shared memory. For neighboring data in our travel direction (x axis), three out of four values can be reused for computation on the next iteration. We can define four private registers to store these values and utilize pipeline copy between registers, reducing by 75% of the global memory access in the x direction. This algorithm has been widely used in seismic 3D finite difference [72], and Figure 4.9 illustrates how it works for the velocity vx computation present in Figure 4.1.



VX computation kernel in thread (ty, tz) is to describe the Register Optimization:

Four registers: Y (Yellow), R (Red), G (Green), B (Blue)

Global memory: vx[i, ty, tz], xx[i, ty, tz] - 3D array (NX+4)*(NY+4)*(NZ+2*pad)

On-chip memory: xy[i, j, k], xz[i, j, k] - One plane Loaded to on-chip L1/Shared memory

1. Preload R=xx[NX+2,ty,tz], G=xx[NX+1, ty, tz], B=xx[NX, ty, tz]
2. For (i=NX+1; i>=2; i--)
    a. load xy, xz to fast on-chip memory
    b. Y=R; R=G; G=B; - pipeline copy between registers, very fast
    c. B = xx[i-2, ty tz]; - only access xx from global memory once in each iteration
    d. VX Computation via Y, R, G, B, xy & xz
3. End for loop and return vx

Figure 4.9: Illustration of register optimization: pipeline register copy to reduce global memory access for asymmetric 13-point stencil computation.

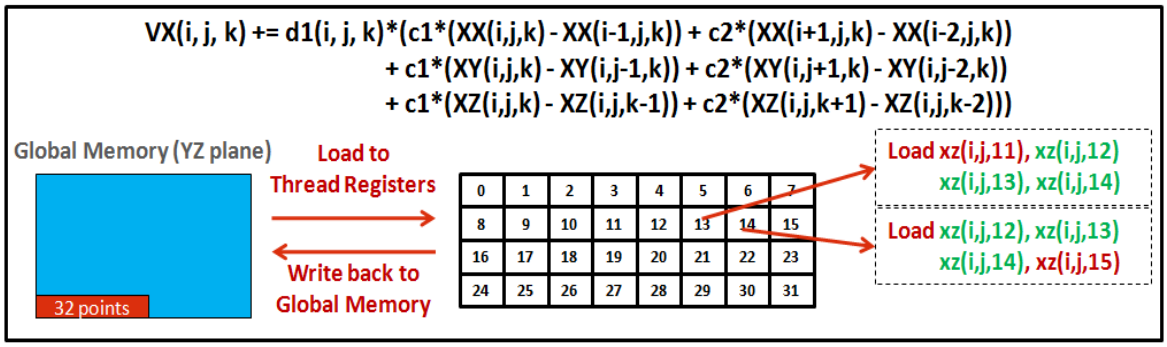


Figure 4.10: Memory access pattern for CUDA warp (size = 32) based on the register optimization algorithm for velocity vx computation, where xy and xz data will be put into the on-chip fast memory and shared by all threads, and xx are stored in private registers.

Velocity vx computation, requires 12 point data access including 4 points from stress xx in the x direction, 4 points from stress xy in the y direction and 4 points from stress xz in the z direction. Based on our decomposition, each CUDA thread will be mapped in y & z directions and compute all the NX data in the x direction. Hence each CUDA warp will be aligned in the z direction and compute the YZ plane. As shown in Figure 4.10, data in the YZ plane are shared by threads inside the same warp, e.g. both thread 13 and thread 14 access xz(i, j, 12), xz(i, j, 13) and xz(i, j, 14), which means 75% of the data are shared between these two threads. Therefore, for each fixed iteration i, we can pre-load stress xy and xz data from the current YZ plane into the on-chip fast memory (shared memory or L1 cache), then all threads in the same warp or block can benefit from the low latency data fetching.

For data in the x direction, each thread accesses its own stress xx and there is no data sharing between threads. For example, the thread (j, k) needs stress xx (i, j, k), (i-1, j, k), (i+1, j, k) and (i-2, j, k), while the thread (j, k-1) needs stress xx (i, j, k-1), (i-1, j, k-1), (i+1, j, k-1) and (i-2, j, k-1), so there is no need to pre-fetch stress xx data into the on-chip fast memory because of the independence. However, there is data sharing for stress xx data between iterations inside the individual thread based on our implementation (Figure 4.9). For each fixed iteration i, stress data xx (i, j, k), (i-1, j, k) and (i+1, j, k) can be re-utilized in the next iteration i+1, while these stress data becomes stress xx (i-1, j, k), (i-2, j, k) and (i, j, k) respectively in the next iteration. Therefore, we do not need to fetch four stress xx data from global memory in each iteration. Instead, we do the pipelining copy between registers first, and then fetch stress xx(i-2, j, k) from the global memory. Register optimization reduced global memory access by three times and improved the data locality effectively.

4.4.5 L1 Cache vs. Shared Memory

GPU Fermi/Kepler provides 64KB on-chip memory and two configuration modes to programmers: 48KB L1 cache and 16KB shared memory for "preferL1", or 16KB L1 and 48KB shared memory for "preferShared". We chose preferL1 over preferShared in our AWP-ODC CUDA implementation because the two layer ghost cells are the bottleneck for high efficiency when fetching data into shared memory. Suppose each GPU block is to compute a plane of size 16x32, the grid size to be fetched into shared memory should be 20x36 due to the 2 layers of ghost cells. Since each block only has 512 threads, some extra data loading operations are required to fetch data in ghost cells. Generally, at least three extra data loads are required to fill the data in ghost cell regions. Hence a load balance issue might occur since not all threads in the block are participating in ghost cell loading, and threads synchronization has to be called before the following computation, which would bring some extra overhead as well. For L1 cache mode, the hardware will take care of the data fetching automatically. As long as the data access pattern in the code is designed to be coalesced (Section 4.4.2 to 4.4.4), we can easily to achieve very high L1 cache hit rate for good data locality.

We also implemented a shared memory version based on Section 4.4.4, which is to pre-fetch one (y, z) plane to shared memory instead of L1 cache in Figure 4.9, and choose preferShared over preferL1 for on-chip memory configuration. However, the time to solution for benchmark size 128x128x960 between the shared memory version and the code implemented in step 4 are almost the same (0.037 sec/per timestep vs. 0.038 sec/per timestep), which means shared memory provides little performance benefit after the first four optimization steps in the AWP-ODC application. Shimokawabe [2] also chose preferL1 over preferShared for a similar stencil-based phase-field simulation for dendritic solidification on Tesla M2050, which is also a 3D stencil computation based on NVIDIA Fermi chipset.

Figure 4.11 shows the improvement in computing FLOPs achieved by these optimizations. The computing performance of our code for size 128x128x960 on Tesla M2090 is 143.8GFLOPs for single precision. We consider this outstanding performance for a memory-bound stencil computation. We compare this performance with the Schafer [73] in which 152.2GFLOPs was achieved with a classical 3D Jacobi Solver for single precision. While Schafer's test was on a slower Fermi device (C2050), our code is more memory intensive, which involves nine 13-point stencil computations and twenty-one 3D variables.

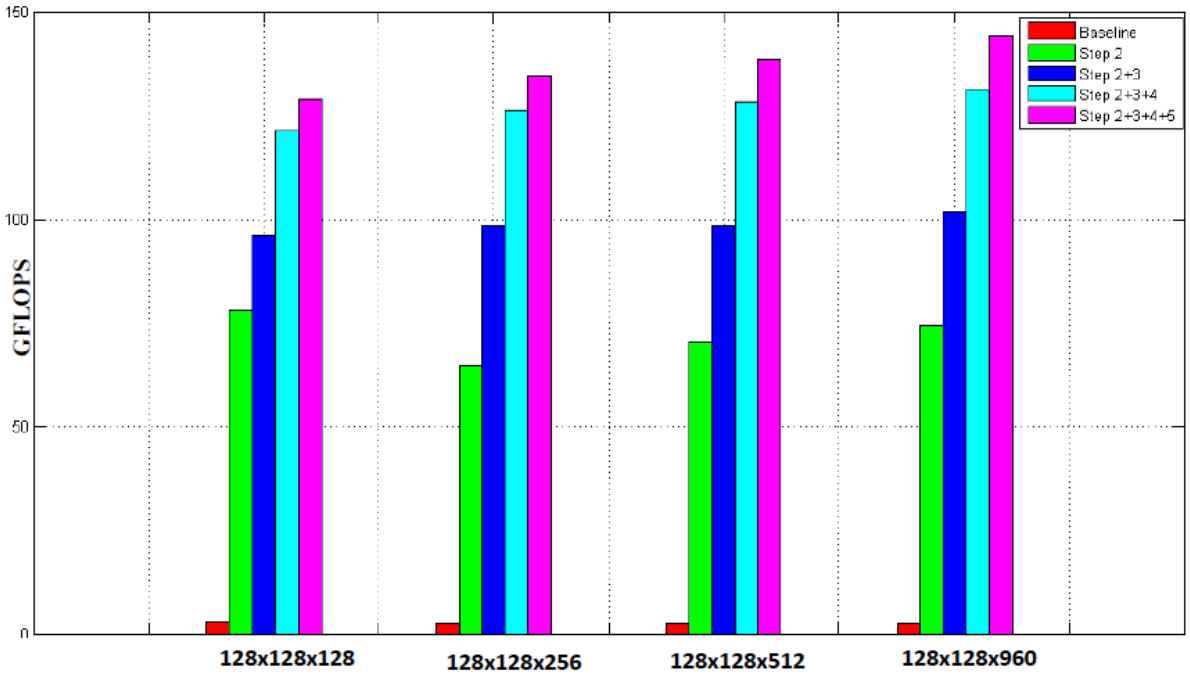
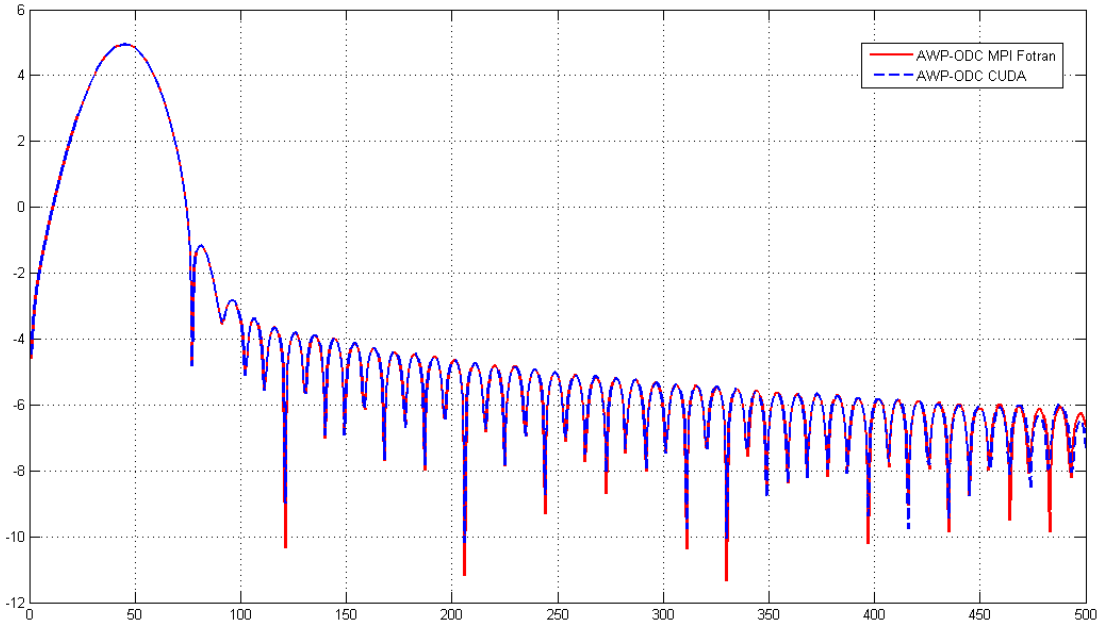


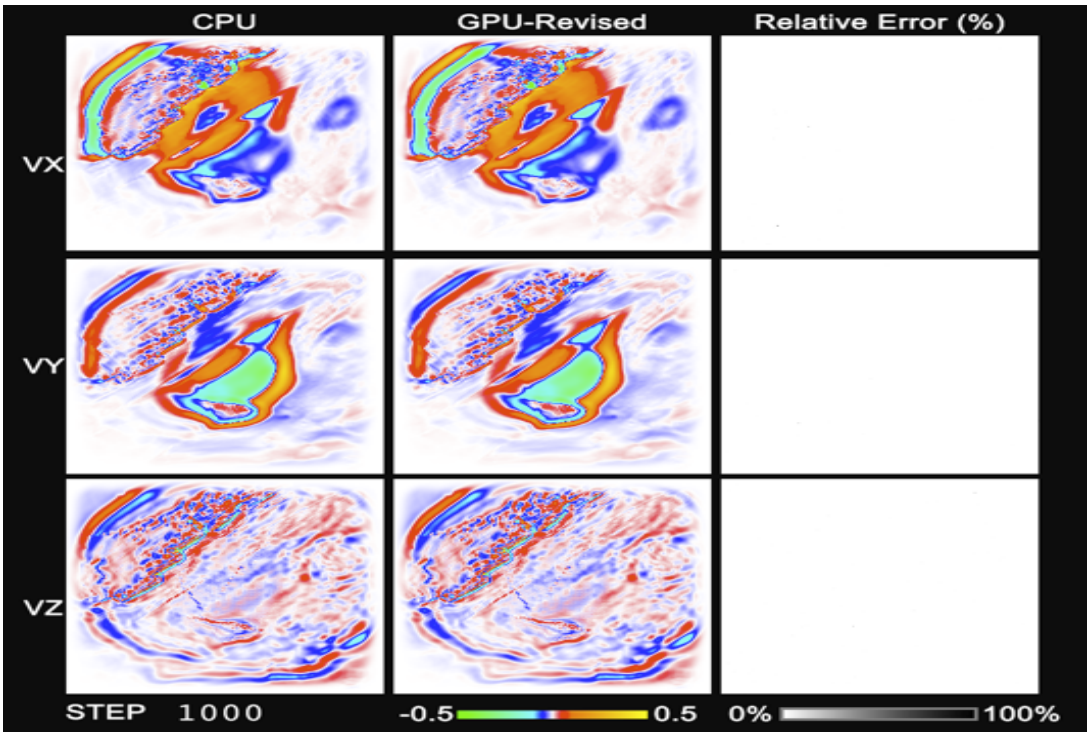
Figure 4.11: Increment of computing GFLOPs achieved by optimization steps for various 3D grid sizes tested on NVIDIA M2090. The baseline version is based on pure global memory implementation without any optimization, and other versions are optimized step by step.

4.5 Experiments and Performance Comparison

The new AWP-ODC CUDA implementation has been carefully validated for correctness. We verified the AWP-ODC CUDA code against the original AWP-ODC Fortran MPI code for production simulations using two different cases. The first one is based on a point source propagated across the whole 3D mesh (128x128x128) with homogeneous mesh data. Figure 4.12(a) is the first 500 timesteps of the log value of the velocity vx recorded at the source point between two codes, showing that the results are almost identical except negligible roundoff differences caused by different programming languages and compilers. The second case uses another point source with a real 3D mesh (non-homogeneous 256x256x256). We generated a 30 second earthquake simulation movie for validation to compare velocity outputs in three dimension. Figure 4.12(b) is a snapshot of the validation movie at timestep = 1000. The third column in the image is the percentage difference between outputs generated by the CPU and GPU codes ((GPU – CPU)/CPU)*100%. The third column is almost blank, which shows the small difference can be neglected and demonstrates the correctness of our AWP-ODC CUDA code.



(a)



(b)

Figure 4.12: (a) The value of log (vx) of the first 500 timesteps, generated by AWP-ODC CUDA and AWP-ODC MPI Fortran, is compared for validation. The 3D grid size for the earthquake simulation is 128x128x128, and the single source point has 91 timestep inputs with homogeneous mesh. (b) The snapshot for a real earthquake simulation movie with non-homogeneous mesh. Errors are shown is column three, which are small enough to be neglected.

To compare the performance between the AWP-ODC CUDA code and the original fully optimized AWP-ODC Fortran MPI code (2010 Gordon Bell Finalist [5]), we chose one multi-core CPU cluster and four NVIDIA GPU devices. The multi-core CPU cluster is the KRAKEN machine located at the National Institute for Computational Sciences (NICS) [49]. The other four GPU devices are NVIDIA Tesla C1060, Tesla C2050, Tesla M2090 and Tesla K20. C1060 is the NVIDIA first generation GPU, while C2050 and M2090 belongs to GPU Fermi, and Tesla K20 is Kepler based architecture. Table 4.3 is a summary of the hardware characteristics of all testbed devices.

Table 4.3: Hardware characteristics of all testbeds: one multi-core CPU cluster and four GPU devices.

CPU/GPU	Concurrency	Frequency	L1 Cache/Shared Memory	Memory Size	Memory Bandwidth
AMD Istanbul on KRAKEN	6 cores	2.6 GHz	64 KB L1	1.33 GB/core	25.6 GB/s
NVIDIA C1060	240 cores	1.3 GHz	16 KB Shared Memory	4 GB in total	102.4 GB/s
NVIDIA C2050	448 cores	1.15 GHz	64 KB L1/Shared Memory	3 GB in total	144 GB/s
NVIDIA M2090	512 cores	1.3 GHz	64 KB L1/Shared Memory	6 GB in total	177 GB/s
NVIDIA Tesla K20	2688 cores	732 MHz	64 KB L1/Shared Memory	6 GB in total	250 GB/s

Figure 4.13 is the performance comparsion between AWP-ODC CUDA code running on the four GPU devices and AWP-ODC Fortran MPI code running on the AMD CPU based multi-core CPU clusters. All benchmark experiments run for 800 timesteps and the measurement is focused on the average time per timestep. For GPU devices, The NVDIA Tesla C1060 is 1.3 capable and the compiler is NVCC 4.0 with options "-O4 -Xptxas -dlcm=ca -maxrregcount=63 -use_fast_math -arch=sm_13". The AWP-ODC CUDA code for Tesla C1060 is a little different from the one implemented in Section 4, and we used shared memory for Step 5 because there is no L1 cache in Tesla C1060. The Tesla C2050 and M2090 are 2.0 capable using the same NVCC 4.0 compiler and the compiler options except "-arch=sm_20". The Tesla K20 is updated to 3.5 compute capability and compiled by the NVCC 5.0 compiler with the same compiler flags except "-arch=sm_35". For the multi-core cluster, we only use a single socket to run the AWP-ODC Fortran MPI code, which means six MPI threads on a single socket of the KRAKEN machine.

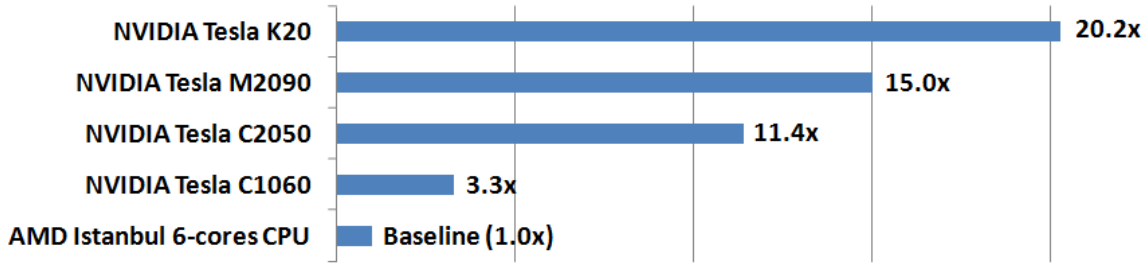


Figure 4.13: Time-to-solution speedup for different GPU generations/architectures. The running time baseline is the AWP-ODC-CPU Fortran/MPI code described in section 4.2 running on AMD Istanbul 6-core CPU at 2.6 GHz with 6 MPI threads for 3D grid size 224x224x1024.

In Figure 4.13, for the same AWP-ODC CUDA code, the average running time per timestep on M2090 is almost 30% less than on C2050, this is because M2090 has two more SMs and higher clock speed. The shared-memory based code running on Tesla C1060 is 4.5x slower than running on M2090, due to the fewer CUDA cores, lack of L1 cache, and lower memory bandwidth. Note that the CUDA code running on Tesla K20 is the fastest, almost 20x faster than the MPI code running on AMD Istanbul 2.6 GHz CPU on KRAKEN with six MPI threads.

Theoretically, K20 has more than 3.5 TFLOPs peak performance in single precision while the M2090 has around 1.3 TFLOPs, so K20 should be 3 times faster than M2090 based on the computing capability comparison. However, our experiments show that the performance on Tesla K20 is only 35% faster than on Tesla M2090 instead of 3x faster. This proves again that the AWP-ODC application is highly memory-bound. The computation performance is partially related with the computing power of the device, but mainly depends on the memory system and data locality. Here are two main reasons for the small performance gain between the Tesla K20 and M2090. First reason is that the global memory size for both K20 and M2090 is the same, which are both 6GB. So even though K20 has more SMXs than M2090, there is not sufficient data for the SMXs to compute and fully utilize their computing efficiency. The other reason is the cache system is not as much improved as the computation. Although the L1 cache/shared memory per SMX has increased 33% to 64KB, and the L2 cache per SMX has also doubled in size (1.5MB), the cache system is shared across the whole SMX. This means 192 CUDA cores in each SMX sharing these new improved L1 and L2 caches in K20, compared to the 32 CUDA cores in each SM on M2090, which are sharing 48KB L1/Shared memory and 768KB L2 cache. Therefore, the L1 and L2 cache size per CUDA core is reduced in K20 compared to M2090, and this might also cause the smaller performance gain.

4.6 Mint Translator and Optimizer

During my single GPU optimization work, I was also working with Dr. Didem Unat and Prof. Scott B. Baden from the Department of Computer Science and Engineering at the University of California San Diego on their Mint project. Here I will breifly introduce the Mint translator and optimizer in this section. Details can be found in our published paper [74].

Mint is a tool developed by Dr. Didem Unat to generate the CUDA code with high computing performance for 3D stencil computation code written in C. It contains two parts: a source to source translator and an optimizer. The source to source translator helps translate the annotated C source code into the CUDA code and the optimizer implements all different levels of optimization for the generated CUDA code. Users of the Mint are not required to be knowledgeable of the GPU architecture or programming, and the use of the Mint is quite straightforward as well. All the work required is just inserting directives between the code regions to be run on the GPU similar to OpenMP, and also putting clear information on the directives, such as source data, destination data, data transfer directions, etc. Then the Mint will do the rest of the work to generate a well optimized CUDA code for direct use without any modification. The performance of the Mint generated code is not as good as manually optimized code, since hand-written code can do more fine tuning. But for general purpose use without aggressive performance requirements, the performance of the Mint code is good enough, and significant effort and time can be saved on performance tuning and optimization.

The AWP-ODC finite difference code has been selected as one of the real-world codes for Mint study. It turns out the Mint achieved 82.6% and 78.6% of the performance of the hand-written and optimized CUDA code on the NVIDIA Tesla C2050 and C1060 respectively [74]. There are some differences between the Mint generated code and the hand-written code causing the performance gap. First, the hand-written code uses texture memory and constant memory as we discussed in section 4.4.1, but Mint does not support either of them. Secondly, the hand-written code uses far fewer registers than the Mint code when implementing the same optimization strategy. This is because a hand coder can reorder instructions and reuse the registers, but the Mint only relies on registers allocated by the NVIDIA compiler NVCC. The last difference is the memory padding method. The Mint code always uses the “cudaMalloc3D” and “cudaPitchedPtr” to pad the 3D array instead of our manually padding method in Section 4.4.3.

4.7 Conclusions and Future Work

We presented implementation and hands-on optimization tuning of AWP-ODC on a single GPU device. The performance of this memory intensive stencil computation code has been significantly improved through effective utilization of GPU on-chip memory and data locality. The single NVIDIA Tesla M2090 GPU benchmark demonstrates sustained performance of 143.8 GFLOPS in single precision for 128x128x960 mesh size, at approximately 10% of the theoretical peak system performance. To our knowledge this is the highest single-GPU performance measured from a seismic application.

Due to the limitation of the algorithm, the AWP-ODC GPU code cannot fully utilize the new features provided by the NVIDIA Kepler architecture, such as the dynamic parallelism and kernel launching kernel. Based on the seismic research requirement, we will reconstruct the algorithm to compute more coarse-grain information in deep regions and more fine-grained in surface areas, instead of a homogenous grid for the whole 3D region. This strategy will not only reduce the memory-intensive load and provide more useful information to the seismologist, but also can let us make full use of these new GPU features to accelerate the code and reduce the time-to-solution. Our goal is always to adopt the latest computing devices or features to maximize the AWP-ODC computation performance incorporated for future petascale or exascale earthquake simulations.

Acknowledgements

Section 4.1 and 4.3-4.5, in part, are a reprint of the material as it appears in the 2012 international conference on Computational Science with title “Hands-on Performance Tuning of 3D Finite Difference Earthquake Simulation on GPU Fermi Chipset” by Jun Zhou, Didem Unat, Dong Ju Choi, Clark C. Guest and Yifeng Cui. Section 4.2, is based on the material as it partly appears in proceedings of the 2010 ACM/IEEE conference on Supercomputing with title “Scalable Earthquake Simulation on Petascale Supercomputers” by Yifeng Cui, Kim B. Olsen, Thomas H. Jordan, Kwangyoon Lee, Jun Zhou, Patrick Small, Daniel Roten, Geoffrey Ely, Dhabaleswar K. Panda, Amit Chourasia, John Levesque, Steven M. Day and Philip Maechling. The dissertation author was one of the primary investigators and author of these two papers.

Chapter 5

Multi-GPU Implementation

This chapter introduces our communication model for AWP-ODC multi-GPU implementation on heterogeneous supercomputers. The communication model has two primary objectives: 1) to reduce the data communication frequency between CPU and GPU inside the computing nodes; and 2) to maximize the overlap time between computation and communication. To better explain our algorithms and strategies, first we present some analysis on the AWP-ODC-CPU communication model running on CPU clusters and discuss why the CPU communication model cannot be used for hybrid CPU-GPU clusters. Then we present details of our communication model and illustrate the purpose of the design. Lastly we provide experiment results running on OLCF Titan, NICS Keeneland and NCSA Blue Waters with excellent linear scalability up to the full machine scale.

This chapter is focused on the data communication solution between nodes on heterogeneous supercomputers. The computation utilizes the same fully optimized GPU code described in Chapter 4. The faster GPU computation code presents more challenges to the communication, since it leaves less computation time for overlapping. Moreover, the communication model discussed in this chapter is not restricted to the AWP-ODC application, but can be extended to any other large-scale 3D stencil computations running on CPU-GPU clusters.

5.1 AWP-ODC-CPU Communication Model

The AWP-ODC-CPU communication model includes two parts: 3D Domain Decomposition and MPI asynchronous communication. For the 3D domain decomposition on CPU clusters, AWP-ODC partitions the simulation volume into smaller sub-domains where the total number of sub-domains matches the number of processors used in the simulation. The whole process is shown in Figure 5.1. Because of the characteristics of the 13-point stencil, the outer surface of each sub-domain requires an extra two-cell padding to correctly propagate waves, and these are called ghost cells.

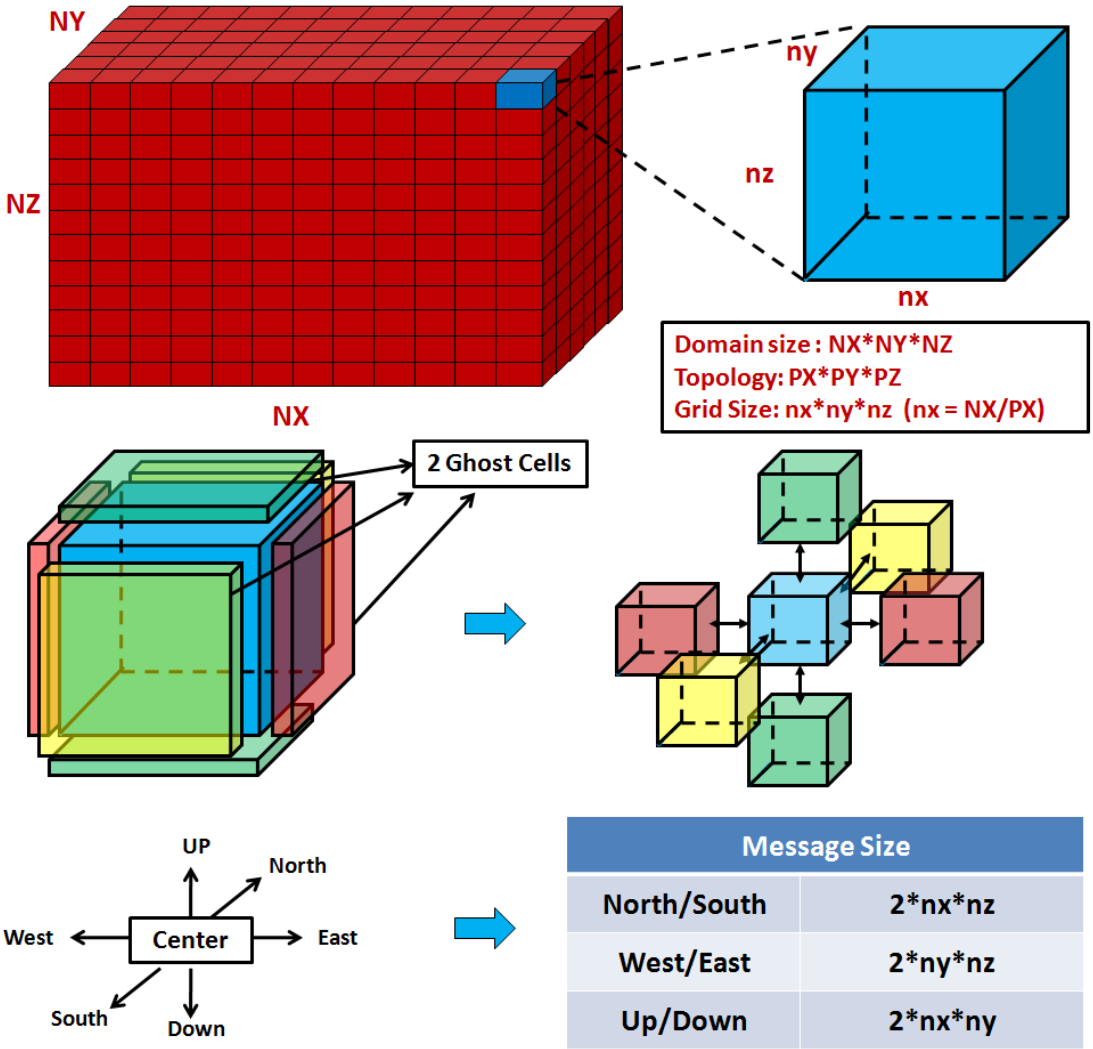


Figure 5.1: shows the 3D domain decomposition process in AWP-ODC-CPU code: each sub-domain has extra 2 ghost cells in each direction with 12 in total. Most sub-domains have to communicate with 6 other sub-domains to exchange data in ghost cell areas during the computation iterations.

The abstract processing of the AWP-ODC-CPU communication model is outlined in Figure 5.2. First, the three velocity components are calculated using the six stress components for the interior and the boundary of the 3D volume. Then velocity values in the boundary volume are exchanged with neighboring sub-domains in six directions including north, south, west, east, up and down. The next six stress components are calculated and communicated in the same manner. All sub-domains in AWP-ODC are 3D grids, thus data along non-contiguous directions must be accumulated in the source sub-domain before being sent out, and disseminated in the destination sub-domain after it is received. These intermediate

data copying stages cannot be avoided, even though we are using the "MPI_SUBARRAY" data type to send data out directly without any memory operation code, these data accumulation and dissemination processes are still implemented in the backend.

AWP-ODC-CPU Main Loop:

Do T= timestep 0 **to** timestep N:
Receiver Buffer Allocation and Initialization (vx, vy, vz, xx, yy, zz, xy, xz, yz)
Compute velocities (vx, vy, vz) using stress (xx, yy, zz, xy, yz, xz)
Sender sends out all velocity data for updating ghost cells in asynchronous way
Wait all communication done and update ghost cells for velocity (vx, vy, vz)
Compute stress (xx, yy, zz, xy, xz, yz) based on velocities (vx, vy, vz)
Sender sends out all stress data for updating ghost cells in asynchronous way
Wait all communication done and update ghost cells for stress (xx, yy, zz, xy, xz, yz)
END DO

Receiver: (vx as an example)

```
MPI_IRecv(VX_Recv_Buffer, North_rank)    //Receive data from north sub-domain
MPI_IRecv(VX_Recv_Buffer, South_rank)
MPI_IRecv(VX_Recv_Buffer, West_rank)
MPI_IRecv(VX_Recv_Buffer, East_rank)
MPI_IRecv(VX_Recv_Buffer, Up_rank)
MPI_IRecv(VX_Recv_Buffer, Down_rank)
```

Sender: (vx as an example)

```
MPI_ISend(VX_Send_Buffer, North_rank)    //Send data to north sub-domain
MPI_ISend(VX_Send_Buffer, South_rank)
MPI_ISend(VX_Send_Buffer, West_rank)
MPI_ISend(VX_Send_Buffer, East_rank)
MPI_ISend(VX_Send_Buffer, Up_rank)
MPI_ISend(VX_Send_Buffer, Down_rank)
```

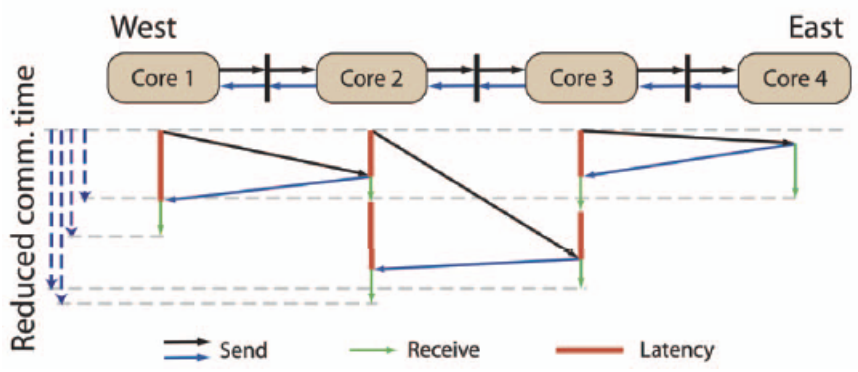


Figure 5.2: shows detailed MPI asynchronous communication process in AWP-ODC-CPU: that significantly reduced the latency caused by the fine-grained MPI Send/Recv order [5].

Most of the communication in AWP-ODC is point-to-point between nearest neighbors, making the performance of AWP-ODC heavily dependent on system interconnect bandwidth. Generally, communication latency between cores in CPU clusters is highly dependent on their physical interconnect distance and the node topology in NUMA systems. Therefore, a sub-domain and its surrounding neighbors that need to communicate with it for data exchange, are grouped and allocated as closely together as possible on the physical machine, ideally inside the same computing node. In addition, to reduce the communication overhead, the AWP-ODC-CPU asynchronous communication model is designed for the CPU-based supercomputers, which allows out-of-order MPI message arrival and unique tags maintain data integrity, resulting in high balanced and low latency communication.

Sreeram et al. from the Department of Computer Science and Engineering at Ohio State University (OSU) improved the MPI communication used in our AWP-ODC model running on homogenous CPU Supercomputers. Based on their research on the Ranger machine located at Texas Advanced Computing Center (TACC) described in Figure 5.3, the computation including both velocity and stress takes up around 63% of the running time, and the MPI communication consumes the other 37% [75]. Because of the different hardware configuration involved, the percentage numbers may vary though not significant. We will use the 63% for computation and 37% for communication in our following analysis.

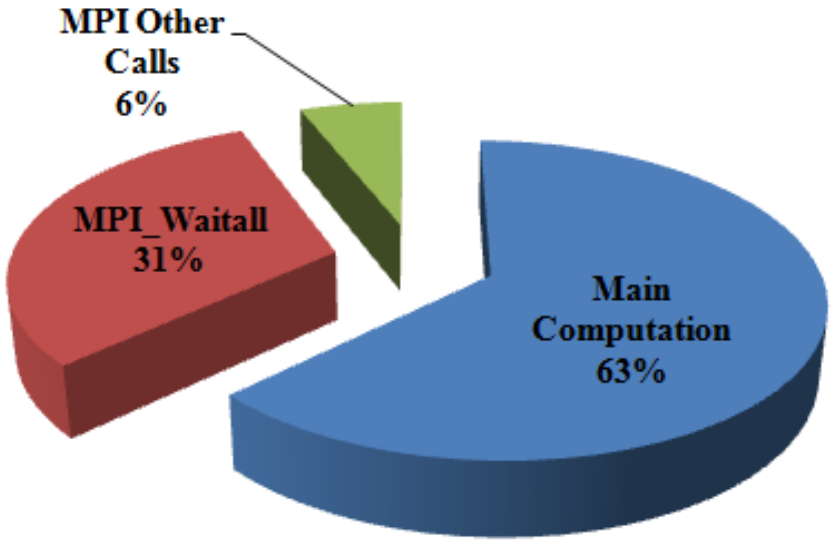


Figure 5.3: Analysis of AWP-ODC-CPU running on homogenous CPU supercomputers: percentages of time spent on computation and communication in the main loop [75].

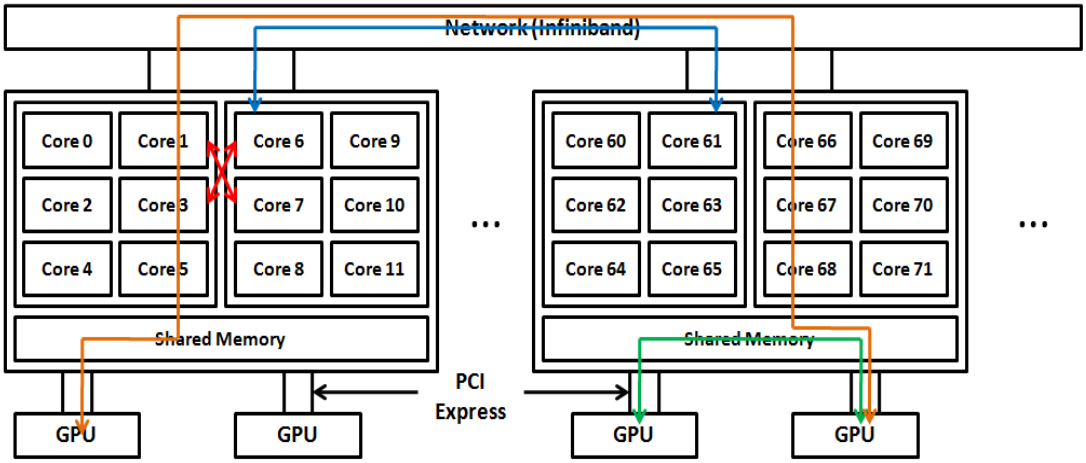


Figure 5.4: Data communication latency model on heterogeneous supercomputers.

As shown in Figure 5.4, the data communication latency is more complex on the hybrid CPU-GPU cluster, including four types of data communication: CPU to CPU intra-node communication (RED: only via shared memory, very fast), GPU to GPU intra-node communication (GREEN: PCI-memory-PCI, slow), CPU to CPU inter-node communication (BLUE: memory-network-memory, slow), and GPU to GPU inter-node communication (ORANGE: PCI-memory-network-memory-PCI, very slow). To quantify the latency time, our definitions list is in Table 5.1:

Table 5.1: Definition List for Different Computation/Communication Patterns.

Mode	Computation/Communication Patterns		Symbol
Computation	Full Computation running on CPU		TC_CC
	Full Computation running on GPU		TC_GG
Intra-Node Communication	**CPU to CPU**	Memory only	TIA_CC
	GPU to GPU	PCI->Memory->PCI	TIA_GG
Inter-Node Communication	**CPU to CPU**	Memory->Network->Memory	TIE_CC
	GPU to GPU	PCI->Memory->Network->Memory->PCI	TIE_GG

For homogenous CPU supercomputers, the computation time is "TC_CC" and the communication latency is dominated by "TIE_CC", so the running time for CPU clusters can be represented as "TC_CC + TIE_CC". For heterogeneous hybrid CPU-GPU supercomputers,

the computation time is “TC_GG” and the communication latency is depending on the longest one “TIE_GG”. Suppose we are using the same AWP-ODC-CPU communication model for our multiple GPU implementation, then the running time for multi-GPUs can be estimated as “TC_GG + TIE_GG”. Based on our results presented in Chapter 4, “TC_GG” is almost 15 times faster than the AMD Istanbul 2.6 GHz CPU (6 cores) on NICS Kraken machine. Thus the percentage of the computation will decrease proportionally. However, the communication time “TIE_GG” consumes significant time than “TIE_CC” as the exchanged data needs to pass though the slow PCI Express 2.0 twice as well as the high speed network. Taking the NICS Kraken supercomputer as an example, the high speed network is a Cray SeaStar 2+ router with bandwidth of 45.6 GB/sec, while PCI Express 2.0 16x attached to the NVIDIA M2090 has only a throughput of 8 GB/sec, which is 5.5 time slower than the network. Because of the two data passes via PCI Express, “TIE_GG” would be 10 times longer than “TIE_CC”.

Assuming to run the same M8 described in the Chapter 3 on CPU-GPU clusters, and considering the same parameter setting (125x125x125 cube per GPU) and communication model, the estimated percentage of running time for computation and communication would result in a heavy imbalance as shown in Figure 5.5. Note that the dramatic performance degradation is due to the bottleneck of the “PCI Express 2.0” bus connection between CPU and GPU.

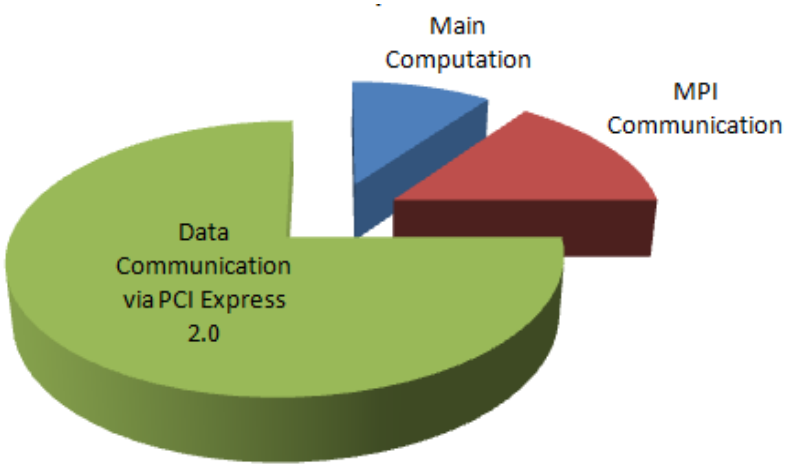


Figure 5.5: Estimated percentages of running time for computation and communication if the AWP-ODC-CPU communication model runs on CPU-GPU clusters.

5.2 AWP-ODC-GPU Communication Model

To better illustrate the AWP-ODC-GPU communication model, we present the four primary parts of our algorithm in detail: two-layer 3D Domain decomposition, communication reduction, in-order MPI communication and effective computation and communication overlapping. This communication model can be adapted to any kind of 3D stencil computation on CPU-GPU clusters. Because of the limitation in AWP-ODC numerical model, our communication reduction can only be reduced once, while other applications with the standard 3D stencil computation could benefit even more, as will be explained in the following sections.

5.2.1 Two-layer 3D Domain Decomposition

As discussed in Section 5.1, the 3D domain is partitioned into many sub-domains and each sub-domain is mapped into a single CPU core for computation. For decomposition on heterogeneous CPU-GPU supercomputer clusters, the first step is the same and each sub-domain is prepared for single GPU computation. The second step is the sub-domain data decomposition inside a GPU for multi-streaming processors. Thus data partitioning must be done twice, and the whole decomposition process is called as "Two-layer 3D Domain Decomposition".

We are using exactly the same optimized GPU code described in Chapter 4 for the multi-GPU implementation. The second "sub-domain data decomposition inside GPU" is identical to what we presented in Section 4.2, which is 2D data decomposition in the Y and Z directions mapped onto GPU SMs. However, for the first step "3D domain decomposition for GPUs", we chose 2D decomposition in X and Y directions instead of 3D in all directions. As shown in Figure 5.6, if the 3D domain is (NX, NY, NZ) and the decomposition topology is (PX, PY, 1), then the sub-domain for each GPU will be (nx, ny, NZ), where the "nx" equals NX/PY and the "ny" equals "NY/PY". In this case, the MPI commutation has been reduced from six directions to four (south, north, west and east), and each GPU is taking the entire Z direction computation. Moreover, larger NZ can make GPU computing more efficient because of the fast Z direction in memory. In summary, the two-layer 3D Domain Decomposition is X/Y partition for GPUs and Y/Z partition for GPU SMs.

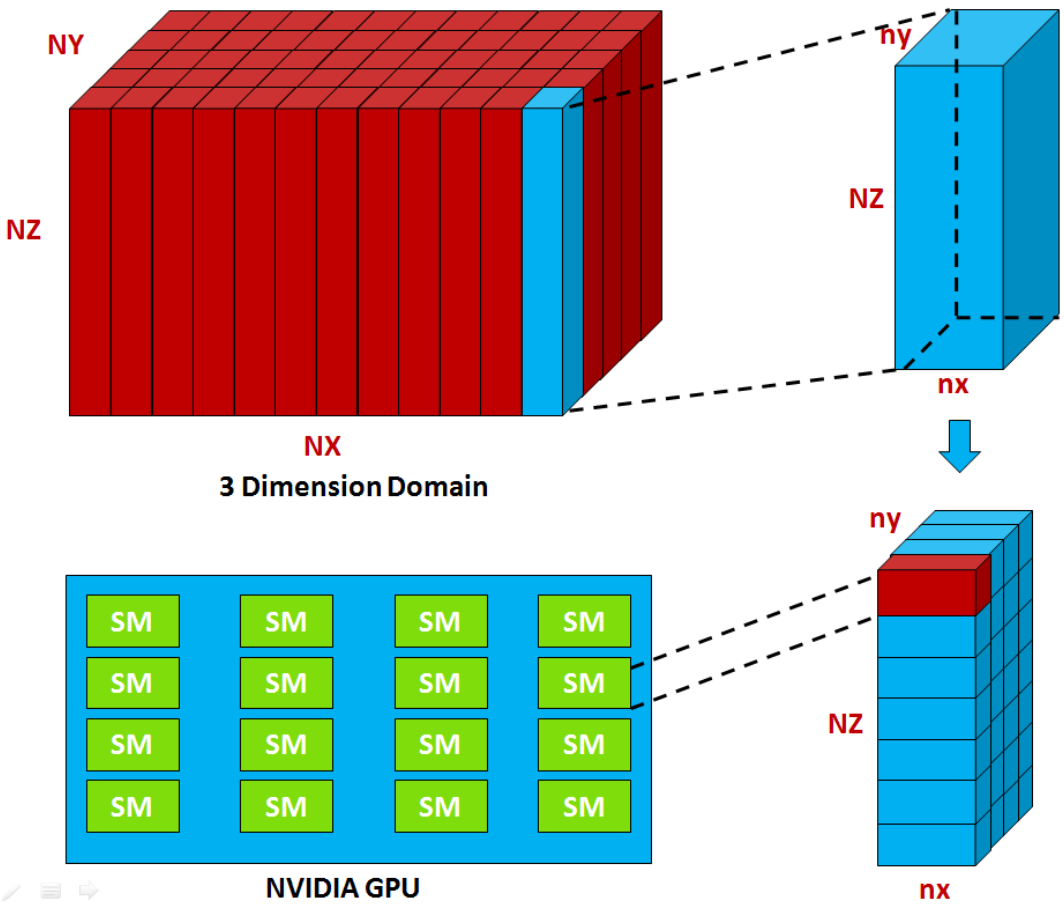


Figure 5.6: Process of Two-layer 3D Domain Decomposition: first step X/Y decomposition for GPUs and second step Y/Z decomposition for GPU SMs.

5.2.2 Communication Reduction

As discussed in Section 5.1, the longest latency communication for the multi-GPU implementation is GPU to GPU between different computing nodes, including passing through the high speed interconnect and the PCI Express 2.0 bus twice. Based on the decomposition described in Figure 5.6 and the original communication algorithm presented in Section 5.1, the MPI message size per variable per GPU becomes "2*ny*NZ" in east and west directions and "2*nx*NZ" in north and south directions. The minimum required GPU to GPU communication frequency is at least eight times per iteration, with four times for velocity data swaps and another four times for stress data swaps. To reduce the communication frequency, our idea is to utilize extra computing workload to replace the communication.

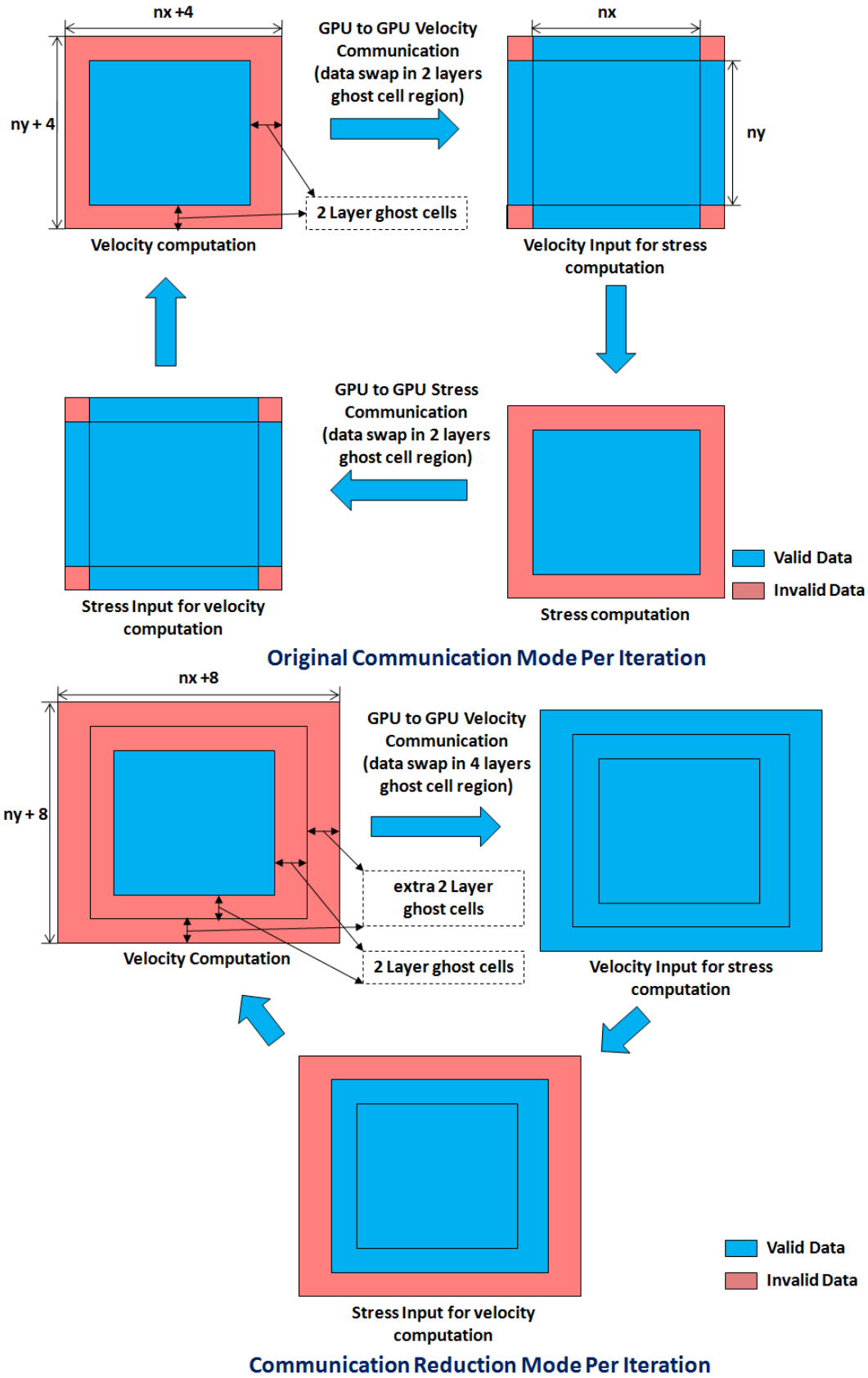


Figure 5.7: Comparison before and after showing communication reduction per iteration.

Since we only have MPI communication in the X/Y directions, and the NZ direction is decomposed inside the GPU, MPI communication only takes place between sub-domains (1: nx, 1: ny, 1: NZ), so we use the 2D X/Y plane instead of 3D sub-domain to show the communication reduction process in figure 5.7.

Because of the nature of 13 point stencil computation, a 2 layer ghost cell region is always required to compute the whole sub-domain. This means for the sub-domain (1: nx, 1: ny, 1: NZ) output, we need (-1: nx+2, -1: ny+2, 1: NZ) input to generate the correct answer. Thus in AWP-ODC, the velocity sub-domain (1: nx, 1: ny, 1: NZ) computation requires stress input (-1: nx+2, -1: ny+2, 1: NZ) and the stress sub-domain (1: nx, 1: ny, 1: NZ) computation also requires velocity input (-1: nx+2, -1: ny+2, 1: NZ) for each iteration. So the original communication plan requires GPU to GPU communication eight times, where four times are for the velocity data swap for the 2 layer ghost cell region (west, east, north and south), and the other four are for the stress data.

Here is the enhanced communication reduction method: for each iteration, first extend an extra 2 layers of ghost cells for velocity (-3: nx+4, -3: ny+4, 1: NZ), though the computation region is still (1: nx, 1: ny, 1: NZ). This means the valid data region is still (1: nx, 1: ny, 1: NZ) after computation. However, when we do the GPU to GPU communication for velocity ghost cells, the data swapping region becomes 4 layers instead of 2, so the valid data region for velocity is (-3: nx+4, -3: ny+4, 1: NZ) after communication. Because of the new velocity valid data region, we have sufficient velocity input information to compute the stress region (-1: nx+2, -1: ny+2, 1: NZ), which will become the stress input to compute the velocity region (1: nx, 1: ny, 1: NZ) in the next iteration. Thus, only velocity data swapping is required and the total number of GPU to GPU communication has been reduced from 8 to 4 for each iteration. Since velocity has 3 components but the stress has 6, the GPU to GPU message size is reduced by a factor of 1/3 because of this reduction method (Table 5.2).

Table 5.2: MPI Message pattern comparison between before and after communication reduction per iteration.

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

This communication reduction not only decreases the GPU to GPU data communication time because of the smaller message size, but also reduces the overhead time caused by data copying between GPU and CPU or MPI initialization. Generally speaking, the time cost for two MPI communications with 2MB data each is longer than one single MPI communication time with 4MB data, which is also the same for data copying between GPU and CPU. Suppose the latency cost is t1 for MPI initialization and t2 for data copying between GPU and CPU, then our communication reduction help save additional "4*t1 + 8*t2" overhead time (4 times MPI communication, plus 4 times data copying from GPU to CPU and another 4 times from CPU back to GPU). Moreover, because of no need for stress data communication, there is more opportunity for overlapping between communication and computation, one of the most important multi-GPU tuning approaches developed.

5.2.3 In-Order MPI Communication

Before dipping into the overlapping algorithm, we must decide the communication mode, including the MPI messaging direction, order and size. For the original communication method before the reduction, there is no message overlap between data in the west/east directions and the north/south directions. Any order is allowed, such as first west/east and then north/south, or first north/south and then west/east (shown in Figure 5.8). However, for the communication reduction scheme introduced, all four corner's information for the extra 2 layers of ghost cell computation is required, that means, another four MPI data swaps are needed, which will bring new overhead and degrade the performance. Alternatively, in-order communication still keep the original four MPI messages. As shown in Figure 5.8 for in-order communication, we first send data in the 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). After the communication is done in the west/east directions, the data in (-3: nx+4, 1: 4, 1: NZ) and (-3: nx+4, ny-3 : ny, 1: NZ) is valid and we can send these data to fill the ghost cell region (-3: nx+4, -3: 0, 1: NZ) and (-3: nx+4, ny+1 : ny + 4, 1: NZ) in the north/south directions. In this case, we keep four MPI messages per iteration, but cover the four corners' information. After the in-order communication, the velocity data in the whole sub-domain (-3: nx+4, -3: ny+4, 1: NZ), including all extended regions and ghost cells is valid as the stress computation input.

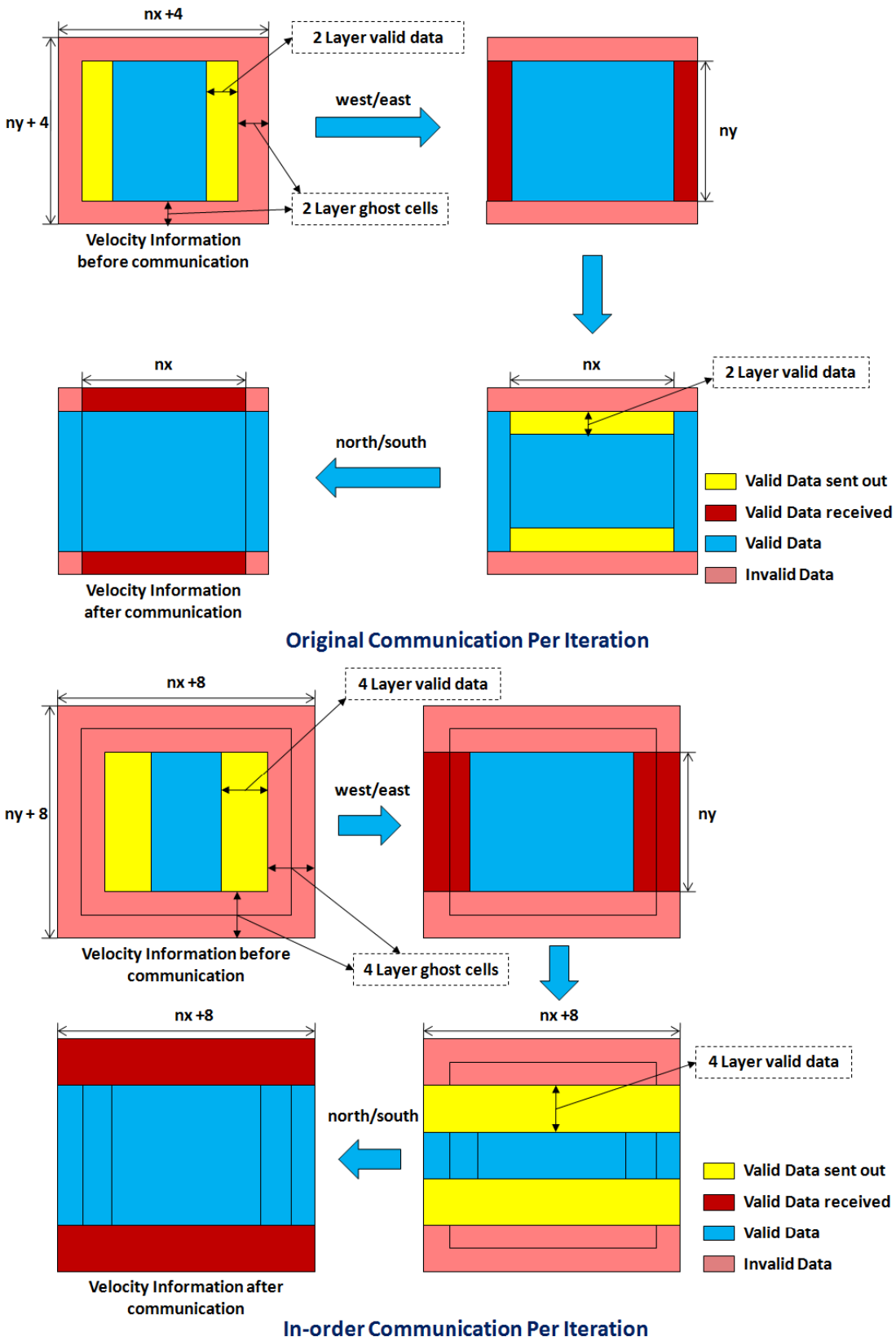


Figure 5.8: Comparison between original communication plan and in-order communication.

5.2.4 Computation/Communication Overlapping

Communication reduction has helped reduce the frequency of data communication and also increased the computing time, while the in-order communication guarantees the accuracy of the data for computation. Next, the crucial point is how to design a good computing/communication schedule to hide the communication effectively.

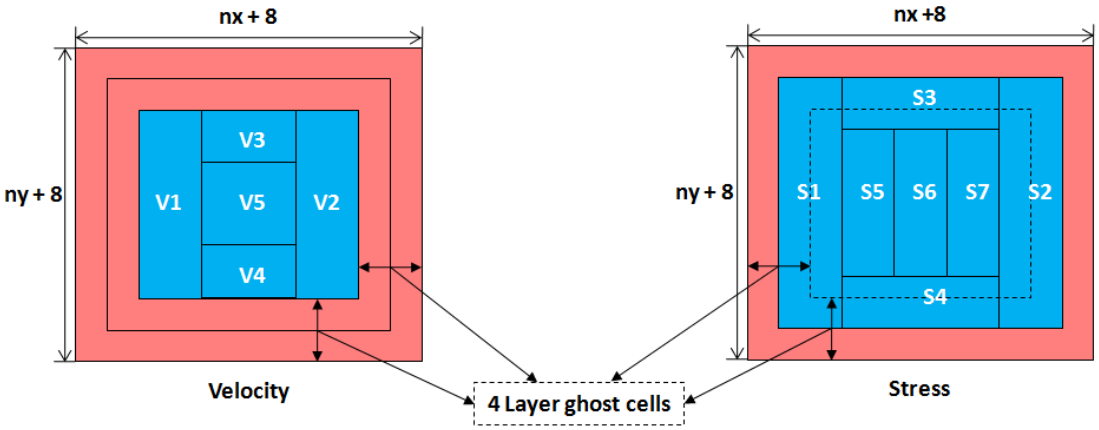


Figure 5.9: Definitions of some symbols for the overlapping algorithm presentation.

Table 5.3: Description of each symbol shown in Figure 5.9 and its corresponding region.

Symbols	Description	Region
Velocity	3D domain with 4 layer ghost cells	(-3: nxt+4, -3: nyt+4, 1: NZ)
V1	Left 4 layers in non-ghost cell region	(1: 4, -3: nyt+4, 1: NZ)
V2	Right 4 layers in non-ghost cell region	(nxt-3: nxt, -3: nyt+4, 1: NZ)
V3	Top 4 layers in non-ghost cells region	(5: nxt-4, 1: 4, 1: NZ)
V4	Bottom 4 layers in non-ghost cells region	(5: nxt-4, nyt-3: nyt, 1: NZ)
V5	The rest non-ghost cell region	(5: nxt-4, 5: nyt-4, 1: NZ)
Stress	3D domain with 4 layer ghost cells	(-3: nxt+4, -3: nyt+4, 1: NZ)
S1	Left 4 layers including 2 ghost cells	(-2: 2, -3: nyt+4, 1: NZ)
S2	Right 4 layers including 2 ghost cells	(nxt-1: nxt+2, -3: nyt+4, 1: NZ)
S3	Top 4 layers including 2 ghost cells	(3: nxt-2, -2: 2, 1: NZ)
S4	Bottom 4 layers including 2 ghost cells	(3: nxt-2, nyt-1: nyt+2, 1: NZ)
S5	1/3 of the rest stress region	(3: nxt/3-1, 3: nyt-2, 1: NZ)
S6	1/3 of the rest stress region	(nxt/3: nxt*2/3-1, 3: nyt-2, 1: NZ)
S7	1/3 of the rest stress region	(nxt*2/3: nxt-2, 3: nyt-2, 1: NZ)

In this new data communication model, only velocity requires data swapping in its four layer ghost cell regions. Our plan is to use the total velocity and stress computation time to overlap all the overhead time caused by MPI communication and data transfer between CPU and GPU via PCI Express 2.0.

```
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 to transfer v1 from GPU to CPU
    Compute v3 for (vx, vy, vz) in GPU and initiate to transfer v2 from GPU to CPU
    Compute v4 for (vx, vy, vz) in GPU and initiate to transfer v3 from GPU to CPU
    Compute v5 for (vx, vy, vz) in GPU and initiate to transfer v4 from GPU to CPU
    Wait for v1/v2 data transferring done and initiate MPI_ISend for v1/v2
    Compute s5 for (xx, yy, zz, xy, yz, xz) in GPU
    Wait for v1/v2 MPI message received and initiate to transfer v1/v2 from CPU to GPU
    Compute s6 for (xx, yy, zz, xy, yz, xz) in GPU
    Wait for v3/v4 data transferring done and initiate MPI_ISend for v3/v4
    Compute s7 for (xx, yy, zz, xy, yz, xz) in GPU
    Wait for v3/v4 MPI message received and initiate to transfer v3/v4 from CPU to GPU
    Compute s1 for (xx, yy, zz, xy, yz, xz) in GPU
    Compute s2 for (xx, yy, zz, xy, yz, xz) in GPU
    Compute s3 for (xx, yy, zz, xy, yz, xz) in GPU
    Compute s4 for (xx, yy, zz, xy, yz, xz) in GPU
END DO
```

Figure 5.10: Effective computation and communication overlapping algorithm for AWP-ODC GPU implementation.

Figure 5.10 presents the flow of our effective computation and communication overlapping algorithm. As illustrated in Figure 5.11, the GPU starts to compute the velocity boundary along the x-axis, V1 and V2, in serial, and copies the data to the CPU immediately by using CUDA asynchronous memory copy when the GPU computation is done. The CPU will send the velocity data V1 and V2 out using the asynchronous "MPI_Isend" when the V1 and V2 data copy is done. During the velocity data V1 and V2 are being processed, the GPU computes the velocity boundary along the y-axis, V3 and V4, simultaneously and also initiates similar asynchronous data copying and MPI data communication after the GPU computation is done. After all velocity boundary computation is completed, the GPU will focus on the remaining inside velocity region V5 computation. Thus the velocity boundary data processing time, which includes data copying time from GPU to CPU and MPI communication time, can be partially overlapped by the velocity computation.

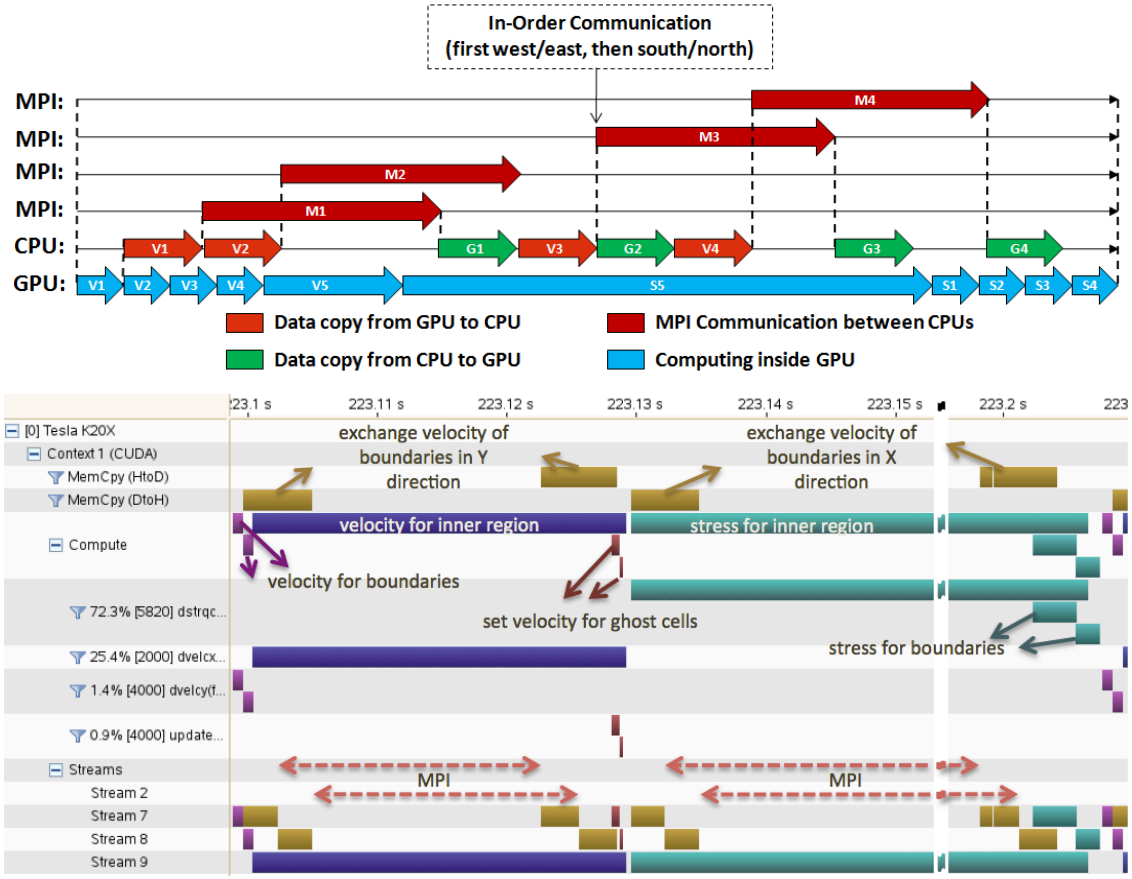


Figure 5.11: Overlap of computation and communication. Top: the concept scheme and Bottom: nvvp profiler output matches well with the design, achieving complete overlap.

Based on the communication reduction, the stress computation region is extended to (-1: nxt+2, -1: nyt+2, 1: NZ) instead of (1: nxt, 1: nyt, 1: NZ), so that no more communication is required for the stress component. In order to increase the overlapping time between communication and computation for better scalability, we divide the stress region into 7 parts (S1 to S7) and reorder the computation sequence from inside to outside. For the inside stress regions S5 to S7, all required information is available for computation if the velocity computation is completed, without need to wait for receiving velocity information before computing the inside stress. The strategy is to compute inside first and then the remaining boundary, so that the MPI communication time for the velocity can also be overlapped by the inside stress computation. Since the barrier function "MPI_Wait" must be called to make sure the boundary information is received successfully, the inside stress region is divided into 3 equal parts (S5, S6 and S7), and one barrier will be inserted respectively between S5/S6 and S6/S7 stress computation initializations. The same data received order (v1, v2, v3 and v4) is

planned, and the asynchronous data copying stream from the CPU back to the GPU will be executed if any velocity data communication for ghost cells finished successfully. Thus the data copying time from CPU to GPU can be also overlapped by the stress computation, plus "multi-layer" overlapping can take place during the iterations. For example, the S6 stress computation inside GPU, V1 data copying from CPU to GPU and the V3/V4 MPI communication between CPUs may be running simultaneously. After the data are copied back from CPU to GPU, there will be no more communication activities while the remaining S1 to S4 is computed inside GPU in sequence. In the end, the updated stress (-1: nxt+2, -1: nyt+2, 1: NZ) is ready as the next iteration input.

5.3 Experiments and Performance Discussion

In this section, we first introduced three key supercomputer systems used for this research: OLCF Titan, NCSA Blue Waters and NICS KIDS system. Then a real earthquake simulation run by both CPU and GPU codes demonstrate validation. Finally, some study results on weak and strong scaling on these supercomputers are discussed.

5.3.1 Introduction of Supercomputer Testbeds

OLCF Titan is the Oak Ridge Leadership Computing Facility's (OLCF) next generation, leadership-class heterogeneous supercomputer [17]. The Titan is upgraded from the Jaguar system, based on the Cray XK7 system with a hybrid CPU-GPU architecture [17]. The Titan Phase5 machine was provided for initial benchmark tests until late 2012, which consists of 960 nodes with 96 nodes per cabin. Each computing node was equipped with one AMD 16 core Opteron^{TM} 6200 series processor and one NVIDIA Tesla X2090 Fermi GPU accelerator. The interconnection between nodes was Gemini, which was a 3-dimensional torus topology providing over 20GB/sec bandwidth per node [17]. The NICS Keeneland Initial Delivery System (KIDS) located at the Georgia Institute of Technology has also been adopted for our research work. It has a similar architecture as 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. Therefore, the Titan Phase5 machine provides 15,360 CPU cores with 960 GPUs [17], while the full KIDS machine provides 1,440 CPU cores with 360 GPUs [24].

The full Titan machine became the No.1 supercomputer in the Top 500 list published in November 2012, which provides a peak theoretical performance of more than 20 PetaFLOPS [17] and was open to public users from early 2013. This giant machine consists of 18,688 physical compute nodes, where each compute node is comprised of one 16-core 2.2GHz AMD Opteron™ 6274 (Interlagos) CPU, one NVIDIA Kepler (K20X) GPU, and 32 GB of RAM. Two nodes share a Gemini™ high-speed interconnect router. These routers are connected in a 3D torus [17]. The Blue Waters machine run by the National Center for Supercomputing Applications (NCSA) at the University of Illinois is another world class supercomputer used for our simulation and benchmarking [76]. The Blue Waters is a hybrid system composed of Cray XE6 and XK7 compute cabinets, where the XE6 computing nodes are pure CPU multi-core architecture, and the XK7 computing nodes are hybrid architecture, including an AMD Interlagos 16-core CPU plus one NVIDIA GK110 Kepler GPU [76]. The Blue Waters provides 11.61 PetaFLOPs theoretical peak performance, but since our GPU code is primarily running on the Cray XK7 hybrid systems, the maximum resource for our use is 3,073 Cray XK7 with 24,576 CPU cores and 3,073 GK110 Kepler GPUs.

5.3.2 Real Small Case Simulation and Validation

Our research is primarily based on OLCF Titan, NCSA Blue Water and NICS KIDS supercomputers. Before the weak scaling benchmarking study on these machines, we first ran a large scale wave propagation simulation of a Mw5.4 Chino Hills, CA earthquake to verify the correctness of the AWP-ODC-GPU code. This simulation has frequency up to 2.5Hz with a mesh dimension size of 1024x1024x1024. The total number of time steps is 75,000, where each timestep simulates a period of 1 millisecond, so the total simulation lasts for 75 seconds. A total of 980 earthquake source points is used for a duration of 2,500 time steps (2.5 seconds in total). AWP-ODC-CPU was previously run on the National Center for Atmospheric Research's (NCAR) Yellowstone [77] homogeneous supercomputer using 512 CPU cores, while AWP-ODC-GPU was run for the same simulation on Keeneland at full system scale (KFS) using 128 NVIDIA Fermi GPUs. Figure 5.12 presents a heat map of the magnitude of the velocity in X direction to compare the two codes. The blank difference images show the accuracy of this new code.

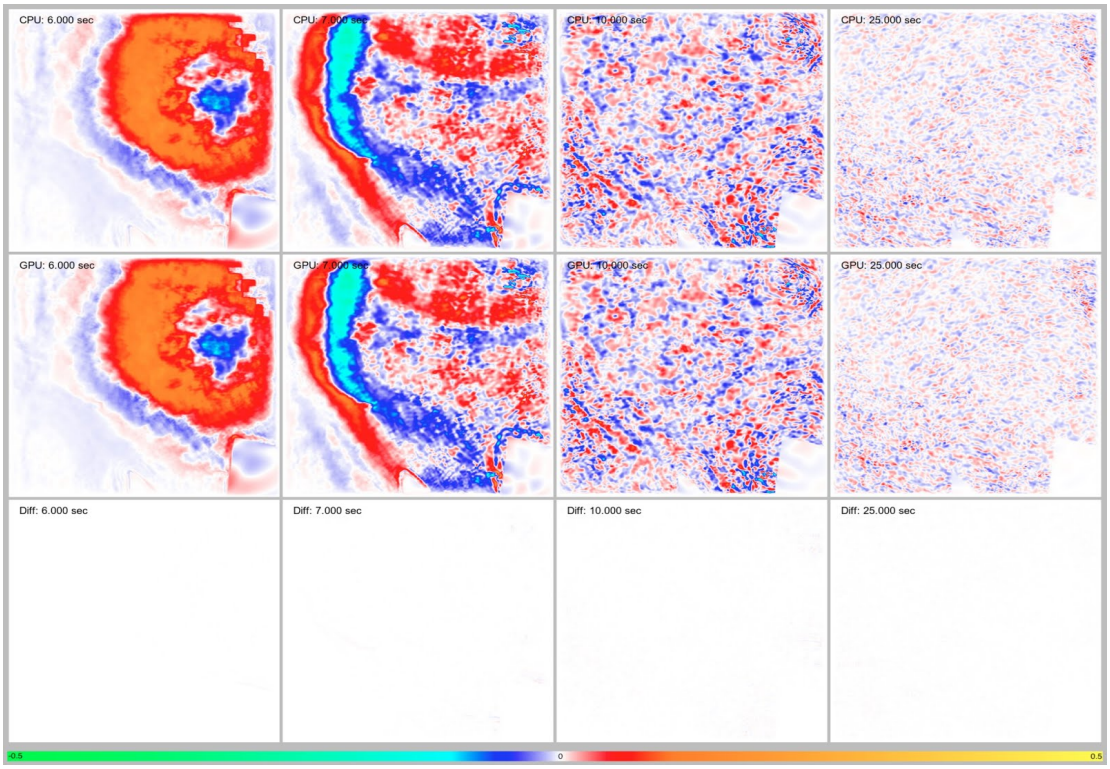


Figure 5.12: Comparison of visualizations of surface velocity in X direction in units of m/s. Four snapshots are taken at seconds 6, 7, 10 and 25 (after 6,000, 7,000, 10,000, 25,000 timesteps respectively). The verification of AWP-ODC-CPU code in this simulation has been performed by geo-scientists for years. The south-east corner of the mesh includes an area with rocks. This uneven distribution of materials in the simulated area results in a difference in the wave propagation compared to other parts of the map.

In collaboration with Kim Olsen of San Diego State University, we have also generated statistical models of seismic velocities and densities in agreement with the results of the analysis of near-surface measurements. The effects of the near-surface heterogeneities on ground motion and scattering are tested using simulations of 0-2.5 Hz wave propagation for the 2008 Mw 5.4 Chino Hills, California earthquake. Figure 5.13 shows the visualization of the earthquake simulation result by our AWP-ODC-GPU code running on NICS Keeneland. The 2008 Mw5.4 Chino Hills earthquake was very well recorded on hundreds of seismic stations. The simulation including the statistical model of the heterogeneities shows several new and interesting results. When compared to strong-motion seismic data from the Chino Hills earthquake, the simulation results tend to predict the duration of ground motion better than the results without the statistical model of the heterogeneities, dependent on the relations for the anelastic attenuation. Details of the visualization can be found at [78].

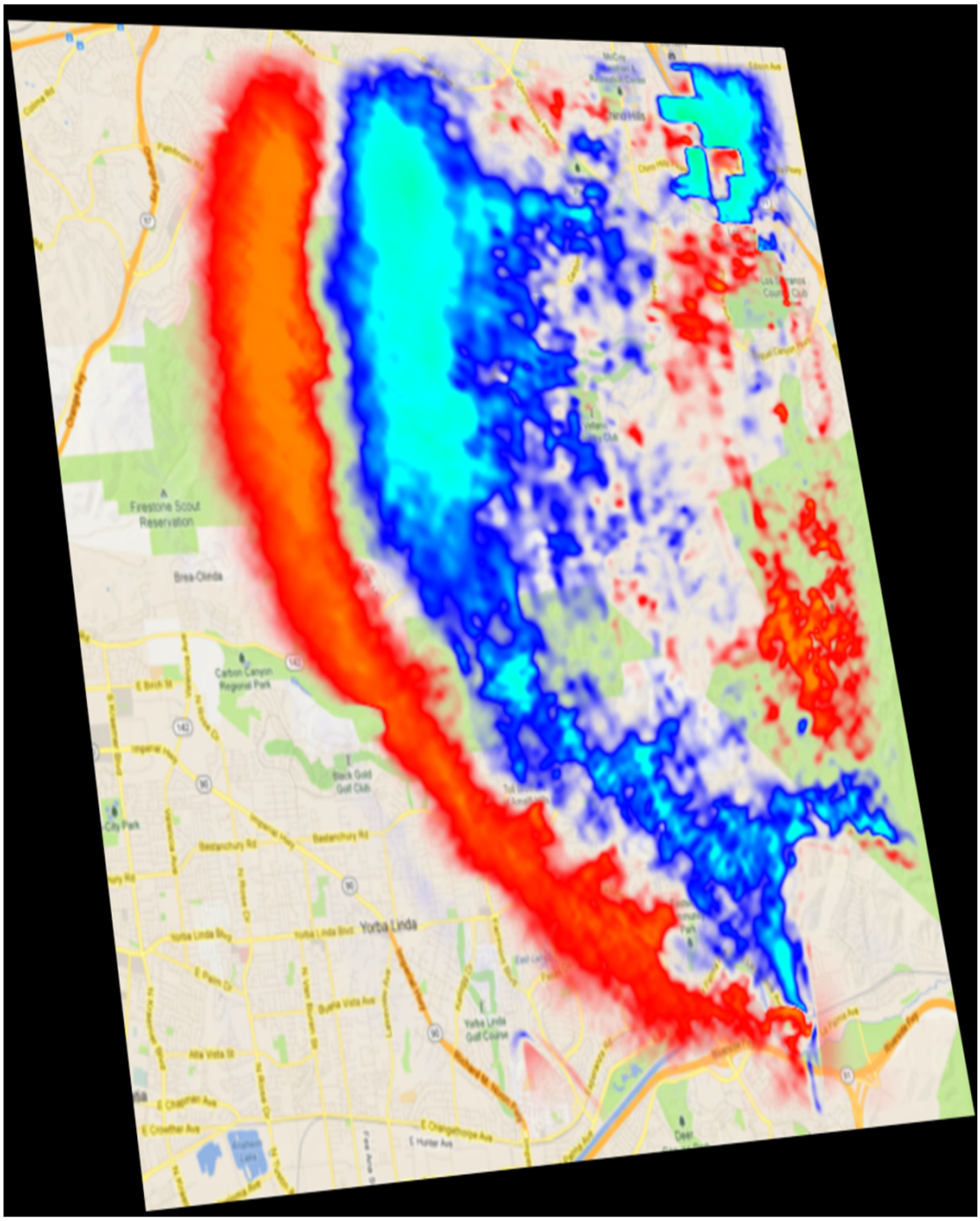


Figure 5.13: 0-2.5 Hz wave propagation of the 2008 Mw 5.4 Chino Hills, California earthquake. Minimum Vs of 200 m/s with a grid spacing of 16 m is calculated using the GPU version of finite-difference time domain code AWP-ODC. A fractal distribution of near surface heterogeneities with a Hurst exponent of 0.1 and σ=10% is added to the near-surface sediments. The simulation results including the statistical model of the heterogeneities predict the duration of ground motion better than the results without the statistical model of the heterogeneities [78].

5.3.3 Study of Strong and Weak Scaling

Based on the introduction of the supercomputer testbeds in Section 5.3.1, there are two kinds of computing resources available for the scaling study. The first one is small clusters based on conventional CPUs plus NVIDIA Fermi GPUs, and the other is new world-class supercomputers installed with conventional CPUs plus NVIDIA Kepler GPUs. Hence separate scaling results are presented here due to the different computing architectures and compiler support.

To compute GFLOPs, the running time performance is measured by the average time spent on one time step after running a benchmark test for 2,000 time steps using the CPU timing function. The number of floating point operations is counted in the code based on 307 FLOP per mesh point per time step discussed in the Section 4.1. Therefore, GFLOPs can be calculated as 307*(the total number of mesh points)/(average running time per timestep). Due to the enhanced overlap algorithm, some communication has been replaced by computation, so the GFLOPs computation must include extended mesh points in the boundary areas. Since different sub-domains might have different GFLOPs value, the total GFLOPs for multi-GPUs is acquired through MPI_AllReduce which gathers the sum of the GFLOPs from all sub-domains. Initialization and output writing are excluded from this calculation. The IO time is negligible when time iterations of tens to hundreds of thousands of time steps are involved.

First, the AWP-ODC-GPU code is benchmarked for the weak scaling study on the Titan Phase 5 system and the Keeneland Initial Delivery System (KIDS) supercomputers, which are based on conventional CPU cores plus NVIDIA Fermi GPUs. Figure 5.14 shows the performance in FLOPS achieved on these 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 the Titan Phase 5 system, but 3 NVIDIA M2090 GPUs per node can be used on the 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 a 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.

Figure 5.14 presents our weak scaling results on these two small scale systems. 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 Fermi GPUs. On the KIDS system, using three GPUs per node achieves triple the GFLOPs compared to one GPU per node. On the Titan Phase 5 system, the peak performance achieved is 101.4 TFLOPs for a mesh size of 7616*6272*1024 (~ 49 billion mesh points) using 952 GPUs, while the peak computing performance achieved on NICS KIDS system is 30.5 TFLOPs for a mesh size of 4032*3360*1024 (~ 14 billion mesh points) using 270 GPUs. Good 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.

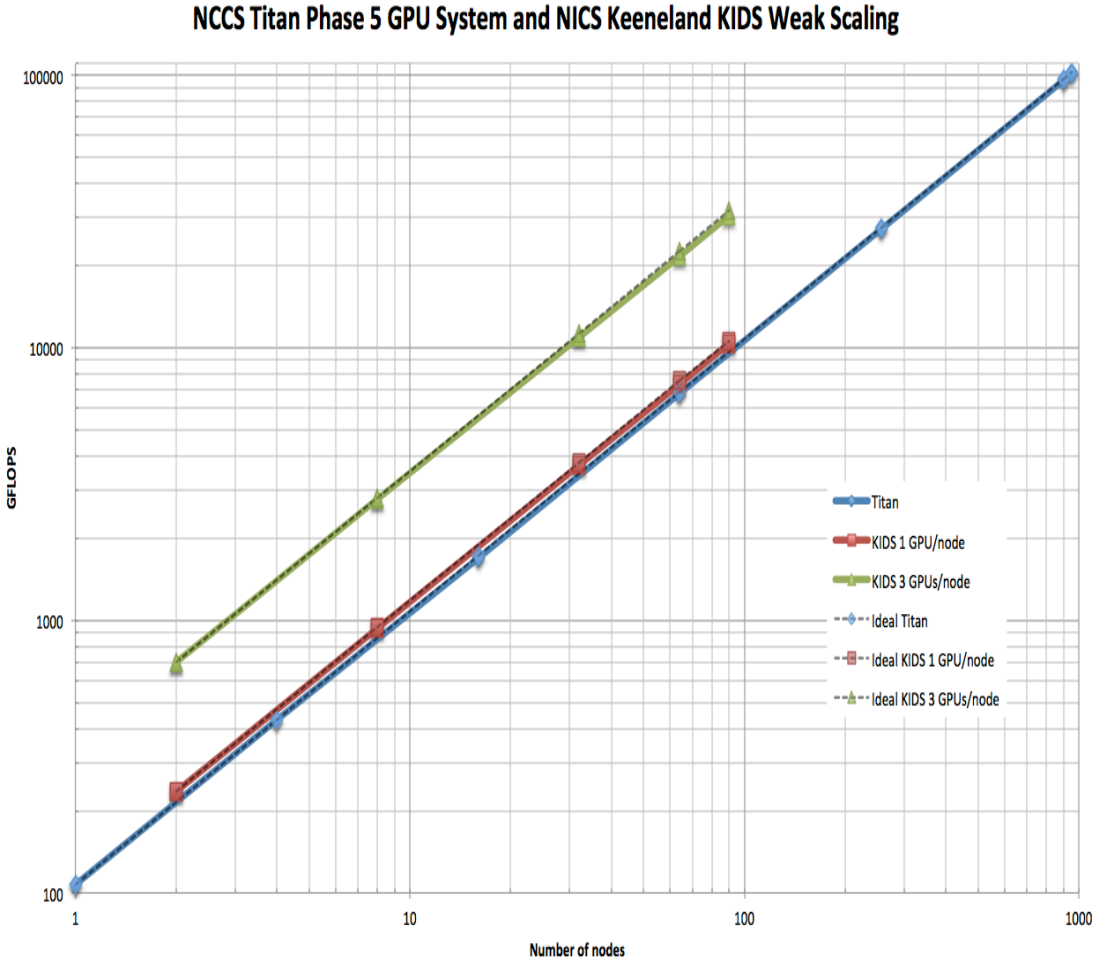


Figure 5.14: Weak scaling study on OLCF Titan Phase 5 system and NICS Keeneland KIDS system in terms of GFlops achieved with respect to the number of nodes requested.

With regard to weak scaling, perfect linear speedup was observed on 90 KIDS nodes equipped with 3 NVIDIA M2090 GPUs per node and 952 Titan Phase5 system nodes equipped with single NVIDIA X2090 Fermi GPU, where 10% of the peak theoretical performance was also achieved. In order to show the AWP code's extraordinary scaling performance, a similar benchmark test was run on the full Titan machine and the Blue Waters machine. In the test, each GPU carries out stencil calculations for a sub-domain with size 160 × 160 × 2048. The total number of points in the domain becomes 160 × 160 × 2048 × N, where N represents the number of GPUs used. Figure 5.15 and Table 5.4 show pefect weak scaling from 16 up to 8192 Titan nodes. To our knowledge, this is a record speedup for a highly memory-bounded scientific problem. A sustained performance estimate of 2.33 PetaFlops on 16,384 Titan GPUs is achieved, which was a 2,000 time-step benchmark run of a problem size of 20,480 × 20,480 × 2,048 or 859 billion mesh points application achieved on Cray XK7.

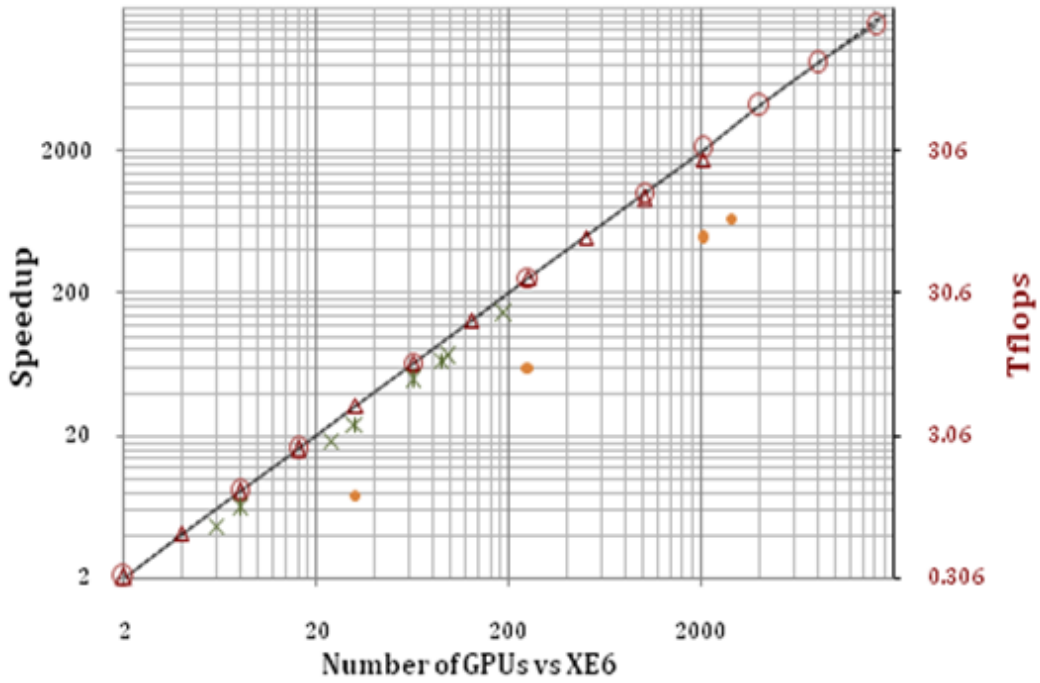


Figure 5.15: Weak scaling and sustained performance using AWP-ODC-GPU in single precision. XK7 exceeds XE6 performance by a factor of 4.2. Solid (dashed) black line is (ideal) speedup on Titan, rounds/triangle/cross points are FLOPS performance on Titan/Blue Waters/Keeneland. Solid round points are FLOPS on Blue Waters XE6. A perfect linear speedup is observed between 16 and 8,192 nodes. A sustained 2.3 Pflop/s performance was recorded on 16,384 Titan nodes.

Table 5.4: Time-to-Solution and Parallel Efficiency on OLEF Titan.

XK 7 Nodes Used	Elements (Millions)	Wall Clock Time	Parallel Efficiency
16 (4x4)	838,860	0.1085	100%
32 (4x8)	1,677,721	0.1084	100%
64 (8x8)	3,355,443	0.1085	100%
128 (8x16)	6,710,886	0.1085	100%
256 (16x16)	13,421,772	0.1085	100%
512 (16x32)	26,843,545	0.1085	100%
1,024 (32x32)	53,687,091	0.1085	100%
2,048 (32x64)	107,374,182	0.1084	100%
4,096 (64x64)	214,748,364	0.1085	100%
8,192 (64x128)	429,496,729	0.1085	100%
16,384 (128x128)	858,993,459	0.1159	93.2%

Notable slowdown was observed in the case of 16,384 nodes, although 93.5% parallel efficiency still can be achieved. Since the application performs only nearest-neighbor communications, continued linear scaling would be expected. The source of this performance degradation is not yet fully understood, but the topology of the network may have played a significant role. One future works is to explore the effect of node topology and evaluate the benefit of topology-aware node placement.

Strong scaling benchmarks were also performed on NCSA Blue Waters and the full OLCF Titan. The small fixed size benchmark was run on Blue Waters whereas others were on Titan. The degradation in performance with an increase in the number of GPUs is expected, as the application becomes bound by communication overhead that arises from less compute work. As the number of GPUs is increased, so does the outer halo region to total sub-volume size ratio in proportion, making the application less effective in overlapping communication and computation.

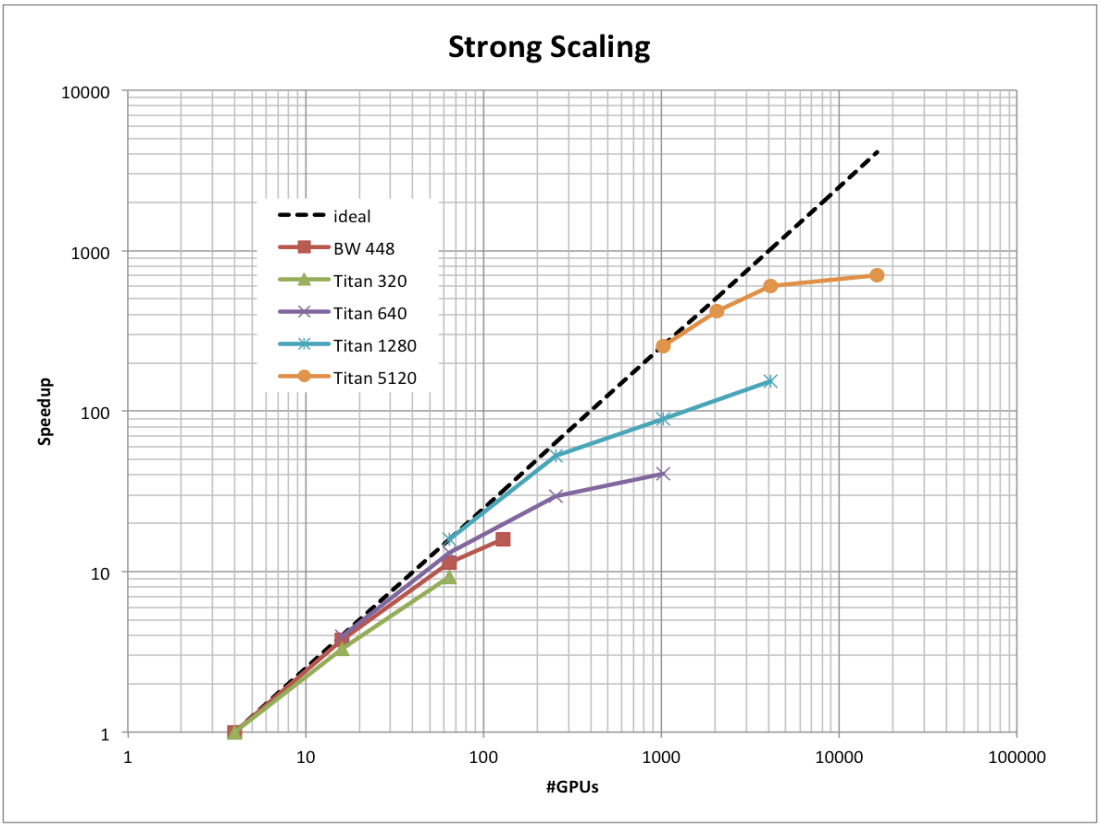


Figure 5.16: Speedup of strong scaling on Cray XT7 at ORNL, with 2D square configuration (Z direction fixed as 2048) for problem sizes of 320, 640, 1280 and 5120.

5.4 Conclusions and Future Work

Recent destructive earthquakes in China, Haiti, Chile, New Zealand, and Japan, highlight the national and international need for improved seismic hazard information. Energy efficient high performance earthquake codes are vitally 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 degradation of the computation performance achieved, a notable communication model has been developed to hide the communication latency, especially to overlap the high latency caused by the data communication between CPU and GPU. With this successful optimization approach, performance studies on OLCF Titan system [17], NICS Keeneland KIDS system [24] and NCSA Blue Waters [76] demonstrated an excellent weak scaling up to the full system. Now

this code has been tested up to the Petascale level and the peak performance we achieved is 2.3 PetaFLOPs. To our knowledge, this is the first Petascale earthquake simulation code in the world at the time of the dissertation writing.

There is still a lot of room for the code optimization. One optimization work is the topology mapping aiming to reduce MPI message latency. When running on OLCF Titan with tens of thousands of GPUs, a good node mapping algorithm can put the communication nodes physically as close as possible so the MPI communication time can be reduced significantly. Other optimization work will focus on a new GPU to GPU communication library. Now the NVIDIA Company provides some functions that support peer-to-peer communications between GPUs, which might already include some system level optimization for supercomputers, and could provide some extra benefits to our code.

Acknowledgements

Section 5.1, is based on the material as it partly appears in proceedings of the 2010 ACM/IEEE conference on Supercomputing with title “Scalable Earthquake Simulation on Petascale Supercomputers” by Yifeng Cui, Kim B. Olsen, Thomas H. Jordan, Kwangyoon Lee, Jun Zhou, Patrick Small, Daniel Roten, Geoffrey Ely, Dhabaleswar K. Panda, Amit Chourasia, John Levesque, Steven M. Day and Philip Maechling. Section 5.2 and 5.3, in part, are a reprint of the material as it appears in the 2013 international conference on Computational Science with title “Multi-GPU Implementation of a 3D Finite Difference Time Domain Earthquake Code on Heterogeneous Supercomputers” by Jun Zhou, Yifeng Cui, Efecan Poyraz, Dong Ju Choi, and Clark C. Guest. Section 5.3, in part, is based on the material as it partly appears in proceedings of the 2013 ACM/IEEE conference on Supercomputing with title “Physics-based Seismic Hazard Analysis on PetaScale Heterogeneous Supercomputers” by Yifeng Cui, Efecan Poyraz, Kim B. Olsen, Jun Zhou, Kyle Withers, Scott Callaghan, Jeff Larkin, Clark C. Guest, Dong Ju Choi, Amit Chourasia, Steven M. Day, Philip Maechling, and Thomas H. Jordan. The dissertation author was one of the primary investigators and author of these three papers.

Chapter 6

Real-World Earthquake Simulation

This chapter presents two real-world earthquake simulations based on our AWP-ODC-GPU code: the first 10Hz deterministic earthquake simulation on the Titan system, and the second CyberShake hazard curve on the NCSA Blue Waters system. These two simulations demonstrate the significant practical and social impact of our AWP-ODC-GPU code, which provides the capability for high-frequency ground motion earthquake simulations (up to 10 Hz), and also helps to save millions of computation hours (up to 500 million) for the future CyberShake 3.0 model.

These two real-world earthquake simulations are big projects led by my co-advisor Prof. Yifeng Cui from SDSC in collaboration with SCEC researchers, including Professor Kim Bak Olsen, Steven Day and their team. My contribution was focused on the implementation of the initial AWP-ODC-GPU for both forward and tensor-valued wavefield codes. The forward code is focus on wave propagation mode for high-frequency ground motion earthquake simulations. The tensor-valued wavefield code targets the CyberShake project for thousands of reciprocal runs to generate the hazard map. These two codes was then added parallel I/O interface similar like our AWP-ODC-CPU implementation, and also integrated for production simulations on Titan and Blue Waters systems. Most of the content and results in this Chapter has been published at the International Conference for High Performance Computing, Networking, Storage and Analysis 2013 (SC 2013) as a technical paper. Simulation outputs have been visualized and adopted by SCEC scientists for further seismic study.

This chapter will go over the simulations in turn. First we introduce the 10-Hz ground motion wave propagation simulation, which is the first time ever in earthquake simulation history by the time of the thesis writing (early 2014). We briefly describe the size and scientific impact of this earthquake simulation and also show the beautiful visualization result. Thereafter we present the CyberShake model and discuss the potential huge computation hours saving benefit from our AWP-ODC-GPU code.

6.1 10-Hz Ground Motion Earthquake Simulation

High-frequency (> 1Hz) deterministic ground motion predictions are critical input to performance-based building design. The accuracy of the simulations is limited by the small-scale complexity of the source and by high-frequency wave scattering in the crust. To investigate this problem, we have simulated high-frequency ground motions on a mesh comprising 443-billion ($20{,}800 \times 10{,}400 \times 2{,}048$) elements in a calculation that includes both small-scale fault geometry and media complexity. This simulation runs on Titan machine using more than 16,000 computing nodes for four hours. Specifically, we have computed the ground motion synthetics using dynamic rupture propagation along a rough fault imbedded in a velocity structure with heterogeneities described by a statistical model. We first carried out simulations of dynamic ruptures using a support operator method [79], in which the assumed fault roughness followed a self-similar fractal distribution with wavelength scales spanning three orders of magnitude, from $\sim 10^2$ m to $\sim 10^5$ m. We then used AWP-ODC-GPU code to propagate the ground motions out to large distances from the fault in a characteristic 1D rock model with and without small-scale heterogeneities. The latter employed the moment-rate time histories from the dynamic rupture simulations as kinematic sources.

Figure 6.1 shows snapshots of the rupture surface wave propagation for crustal models with and without the media heterogeneities. The fractal roughness is controlled by a Hurst number, which we set at 0.2, and the size of the heterogeneity by a standard deviation, which we set at 5%, as constrained by near-surface and borehole velocity data. Note how the wave field in the bottom snapshot is scattered the small-scale heterogeneities, which generates realistic high-frequency synthetics. A few seismograms are shown to compare models with and without the small-scale structure.

The earthquake simulation results show realistic features. The acceleration spectra from the simulation are nearly flat up to almost 10 Hz, in agreement with theoretical predictions. Moreover, the simulated response spectra compare favorably with spectra obtained from the empirical ground motion prediction equations (GMPEs) currently used by building engineers, which are calibrated to high-frequency recordings of earthquake ground motions. This is the first 10-Hz ground motion earthquake simulation in seismic science research and the output again proves the accuracy of the AWP-ODC-GPU code.

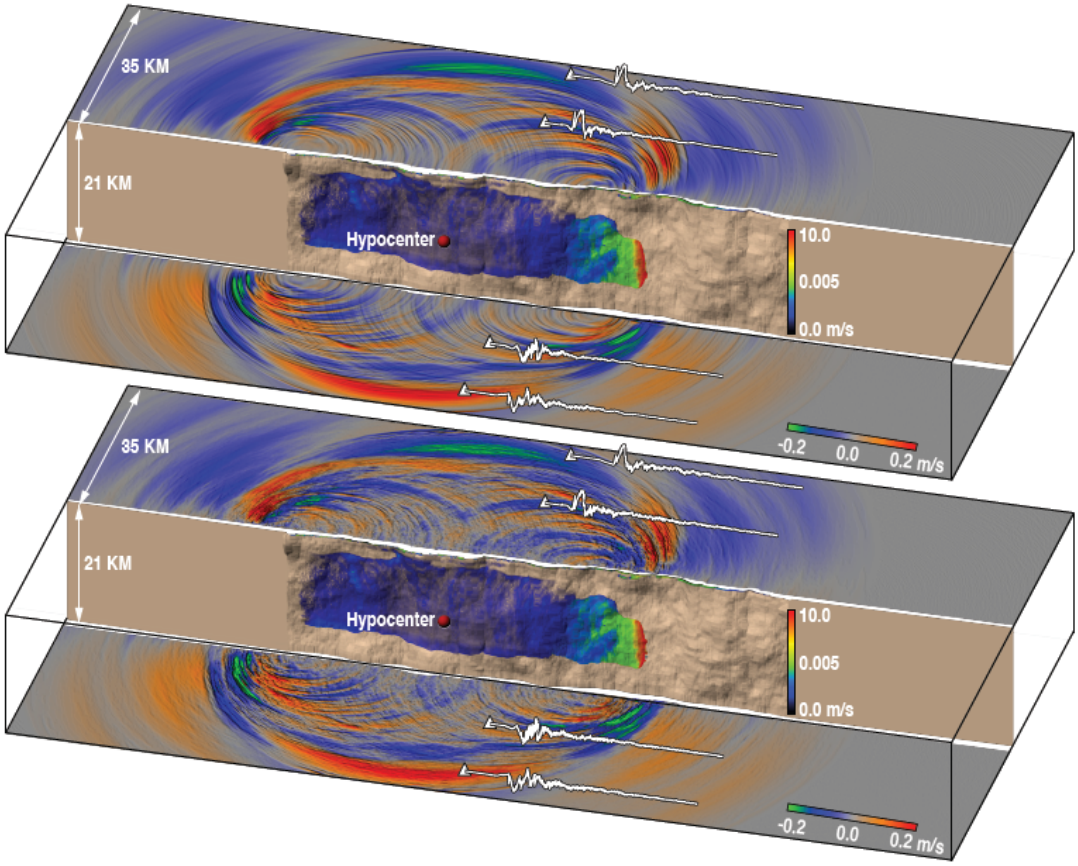


Figure 6.1: Snapshots of 10-Hz rupture propagation and surface wavefield for a crustal model without (top) with (bottom) a statistical model of small-scale heterogeneities. The displayed geometrical complexities on the fault were included in the rupture simulation. The associated synthetic strike-parallel component seismograms are superimposed as while traces on the surface at selected sites. The part of the crustal model located in front of the fault has been lowered for a better view. Note the strongly scattered wavefield in the bottom snapshot due to the small-scale Heterogeneities.

This simulation has a domain size of 416 km×208 km×41 km with a spatial resolution of 20 meters. The size of this run is slightly larger than the record M8 San Andreas fault simulation [5]. The run took only 5 hours and 30 minutes to complete 170 seconds of simulation time whereas M8 ran on approximately 220K CPU cores for 24 hours. We emphasize that this simulation included 6.8 TB of input and 170 GB of output. To our best knowledge, this is the first sustained PetaFLOP seismic production simulation to date, and a new record for earthquake simulation in terms of scale. These results are particularly remarkable considering that memory-bounded stencil computations typically achieve a low fraction of theoretical peak performance.

6.2 CyberShake Hazard Model and Simulation

Probabilistic seismic hazard analysis (PSHA) has been effective in helping decision-maker reduce seismic risk and increase community resilience. The arrival of Petascale computing has opened the door to full-scale, physic-based PSHA. For example, earthquake simulations have recently been validated against ground motions recorded up to 4 Hz, with promising results [80], and we are pushing these comparisons to even higher frequencies. However, in order to calculate seismic hazards in California and other tectonically active regions, simulating just a few earthquakes won't do; we must adequately sample earthquake distributions from probabilistic models, such as the Uniform California Earthquake Rupture Forecast (UCERF) [81]. Using standard "forward" simulation methods, computing three-component seismograms from M sources at N sites requires M simulations. For the UCERF model in Southern California, M> 10^5; i.e., hundreds of thousands to millions of possible earthquake sources must be modeled, which cannot be done directly, even at petascale.

To overcome this scale limitation, SCEC has built a special simulation platform, CyberShake, which uses the time-reversal physics of seismic reciprocity to turn the problem around [82]. A complete tensor-valued wavefield (the strain Green tensor or SGT) is calculated for a system of point forces at surface sites; seismic reciprocity then allows us to compute seismograms at those sites by fast (embarrassingly parallel) quadratures of the SGT over the fault surfaces. This "reciprocal" simulation method can generate 3-component seismograms for M sources at N sites with only 3N simulations. For the Los Angeles region, the near-surface geologic structure can be interpolated to produce high-resolution seismic hazard maps with N as small as 200-250, reducing the computations by a factor of 2,000. Scientific workflow software is used to manage the hundreds of millions of jobs needed to populate a CyberShake model [83].

Using the CyberShake platform, SCEC have created the first physics-based PSHA models of the Los Angeles region from suites of simulations comprising ~10^8 seismograms. These models are "layered", allowing earthquake engineers and other users to access ensembles of hazard curves (representing epistemic uncertainties), to disaggregate the calculations and identify the ruptures that dominate the hazard at a particular site, and to retrieve the actual seismograms, which can then be used to drive full-physics engineering models. Figure 6.2 shows the detailed layering information of the CyberShake hazard model.

CyberShake brings the computational challenges of physics-based PSHA into sharp focus. The current models are limited to low seismic frequencies (≤ 0.5 Hz). Goals are to increase this limit to above 1 Hz and produce a California-wide CyberShake model using the new UCERF3 rupture forecast, which is scheduled to be released this year. The computational size of the statewide model will be more than 100 times larger than the current Los Angeles models. Our progress towards exascale is also being driven by the application of full-3D waveform tomography to the development of the seismic velocity models [84-85], which are required as input to CyberShake. Full-3D tomography using 3-component seismograms from M sources observed at N stations requires at least 3N + Mwavefield simulations per iteration [86].

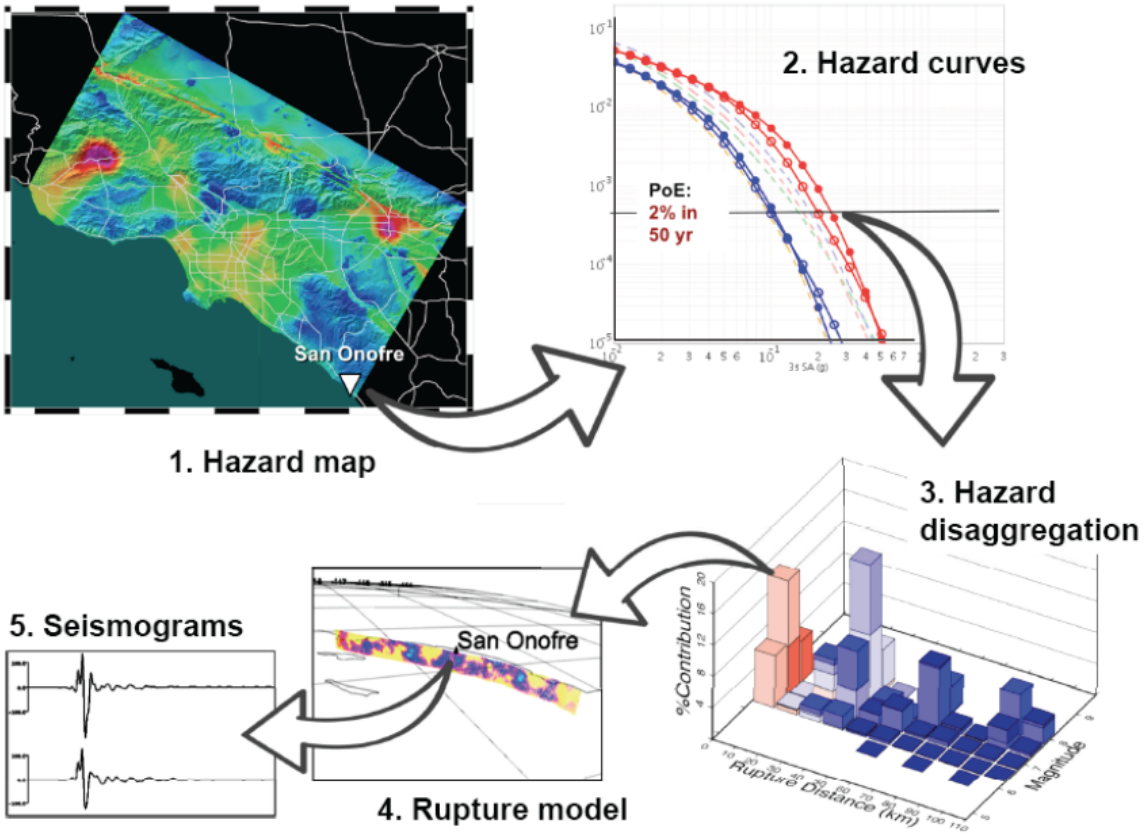


Figure 6.2: The CyberShake hazard model, showing the layering of information. (1) Hazard map for the LA region (hot colors are high hazard). (2) Hazard curves for a site near the San Onofre Nuclear Generating Station. (3) Disaggregation of hazard in terms of magnitude and distance. (4) Rupture with the highest hazard at the site (a nearby offshore fault). (5) Seismograms simulated for this rupture. Arrows show how users can query the model starting at high levels (e.g. hazard map) to access information of progressively lower levels (e.g. seismograms) [87].

PSHA results are typically delivered by hazard curves, which relate ground motion on the X-axis to probability of exceeding that level of ground motion on the Y-axis, for a site of interest. To verify AWP-SGT, we calculated a CyberShake hazard curve using the GPU version of AWP-ODC-SGT, and compared it to a hazard curve using the CPU version; the two are numerically almost identical (shown in Figure 6.3). Calculation of a hazard curve involves SGT time series data from over half a million locations in the volume, providing rigorous verification.

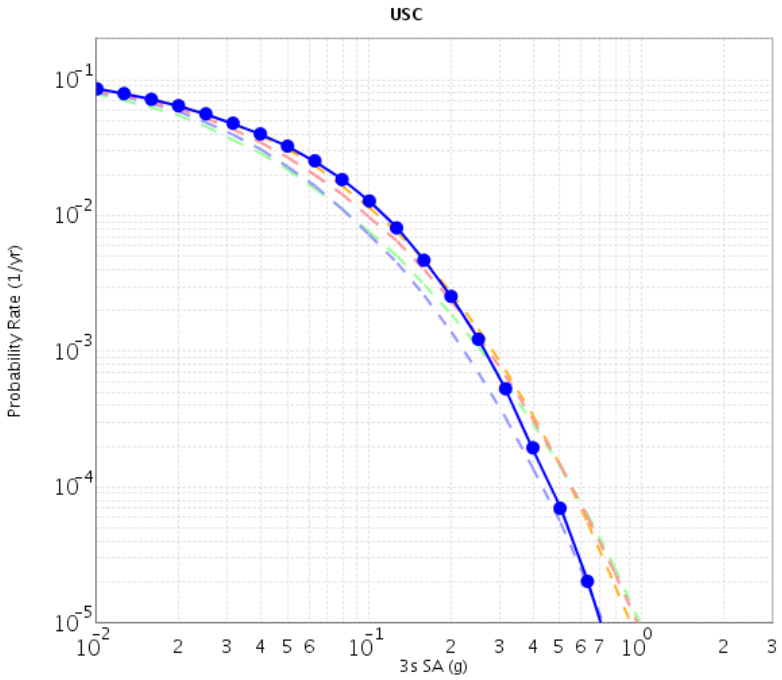


Figure 6.3: PSHA hazard curve calculated for the University of Southern California (USC) site. The horizontal axis represents ground motion at 3 seconds spectral acceleration, in terms of g (acceleration due to gravity). The vertical axis gives the probability of exceeding that level of ground motion. The blue line is the curve calculated using CyberShake with AWP-SGT GPU code. The dashed lines are hazard curves calculated using four common attenuation relationships which provide validation of the CyberShake methodology [87].

One of the primary motivations of implementing AWP-ODC-GPU code is to accelerate CyberShake calculations. We are planning to use CyberShake to calculate a state-wide seismic hazard map using 3D waveform modeling that will improve the earthquake shaking history forecasts and help engineers design safer buildings and retrofit existing high-risk buildings. When using the heavily optimized CPU code AWP-ODC, it is expected to require 662 million allocation hours to complete a CA state-wide hazard map at a maximum frequency of 1-Hz. Our new AWP-SGT GPU code running on Cray XK7 demonstrates a performance improvement of a

factor of 3.7 compared to the CPU code running on Cray XE6. Table 6.1 provides some detailed comparisons of calculating SGTs on XK7 versus XE6, and demonstrates the saving of 579 millions of allocation hours when using the accelerated (CPU+GPU) AWP.

Table 6.1: CyberShake 3.0 Strain Green Tensor Calculations: 1) XE6 node (dual Interlagos); 2) XK7 (Operaton + Kepler K20X); 3) Wall clock time based on measurements on Cray XE6/XK7 at NCSA for two Strain Green Tensor calculations per site; 4) Based on total 5000 sites required for the generation of California state-wide seismic hazard map at a maximum frequency resolution of 1-Hz; 5) CPU + GPU saving counts the use of XK7 CPUs for post-processing of seismogram extraction as co-scheduling, involving 6.2 million rupture variations calculations per site.

CyberShake 3.0	**CPU[1] only**	**GPU[2] only**	**CPU + GPU[2]**
XE6[1]/XK7[2] nodes	400	400	400
WCT[3] per site	10.36hr	2.80 hr	2.80 hr
Total SUs charged[4]	662 M	168 M	168 M
Saved in Million SU[5]		495 M	**579 M**

The results show the capability of our AWP-ODC-SGT GPU code again, which can help to serve as the main computational engine for the CyberShake calculation. As shown in the table 6.1, the use of the AWP-SGT GPU-only code is expected to save up to 500 million hours of computation required for the proposed statewide CyberShake 3.0 model, in addition to reducing dramatically the time-to-solution. This AWP-ODC-SGT GPU code will provide highly scalable solutions for other problems of interest to SCEC as well as the wider scientific community, including full-3D waveform inversions to obtain better velocity models for use in structural studies of the Earth across a range of geographic scales.

Acknowledgements

Chapter 6, in part, is based on the material as it partly appears in proceedings of the 2013 ACM/IEEE conference on Supercomputing with title “Physics-based Seismic Hazard Analysis on PetaScale Heterogeneous Supercomputers” by Yifeng Cui, Efecan Poyraz, Kim B. Olsen, Jun Zhou, Kyle Withers, Scott Callaghan, Jeff Larkin, Clark C. Guest, Dong Ju Choi, Amit Chourasia, Steven M. Day, Philip Maechling, and Thomas H. Jordan. The dissertation author was one of the primary investigators and author of this paper.

Chapter 7

Future Work and Conclusions

In this chapter, we will discuss the future high performance seismic simulation on the upcoming computing architectures and also summarize the full dissertation.

7.1 Seismic Simulation Model

The current seismic simulation model is based on 3D structured mesh where mesh points are equally distributed across the 3D domain. However, the wave propogation on the surface is the most critical result for generating hazard map for scientific research or industry purpose. Wave propogation information inside the 3D domain, especially far away from the surface, is less important. Therefore, the seismic simulation model can be improved as coarse-grain for 3D domain and fine-grain for 2D surface simulation, which means there will be more mesh points on the 2D surface or near the surface and less mesh points (shown in Figure 7.1). The purpose of this simulation model improvement is to heavily reduce the computation loads while achieving the same output quality.

As described in section 2.1, NVIDIA GPU with Kepler architecture supports dynamic parallelism, allowing GPU kernel to launch its child kernel. The proposed seismic simulation model can gain significant benefits from this GPU feature. In each timestep, the main GPU kernel is computing coarse-grain for the whole 3D domain. If the mesh point is close to the surface, it will launch a child kernel to compute fine-grain for the 2D surface. After all child kernels return, it iterates to the next timestep. In addition, due to the coarse-grain computation for the 3D domain, the computing performance would gain more speedup because of the less memory-bound. The main difficulty of this proposed seismic simulation model is the data processing on the interface between fine-grain and coarse-grain. This is because some wave reflection might be generated between this interface as a result of the different density of the mesh points between two domains. This issue is being investigated by computational scientists in our lab and would be done in the near future.

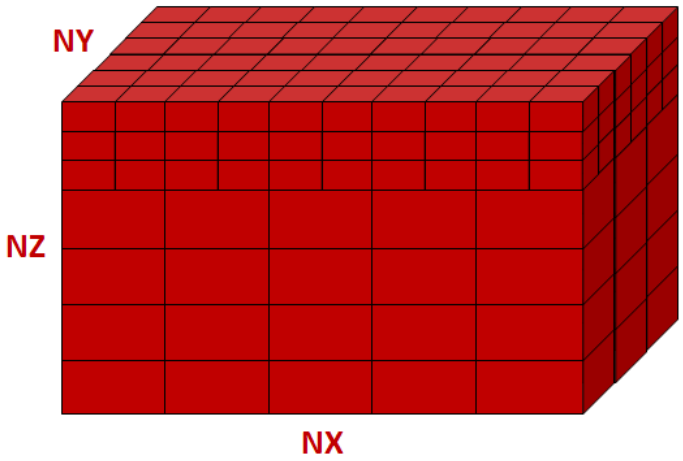


Figure 7.1: proposed seismic simulation model: coarse-grain computation for full 3D domain and fine-grain computation for 2D surface/near surface domain.

7.2 Intel MIC Architecture

The Intel MIC (Many Integrated Core) architecture is developed to compete with NVIDIA Tesla GPU for HPC market (See Figure 7.2). Each MIC co-processor has more than 50 x86 cores and each core allows four hardware threads running simultaneously with 512-bit wide SIMD instructions. As mentioned in the section 2.1, the No.1 supercomputer Tianhe-2 machine based on the Top500 list released in Nov. 2013 is installed with such architecture, while each computing node of Tianhe-2 contains two Intel Ivy Bridge Xeon processors and three Xeon Phi processors (MIC architecture). Similar to GPU accelerators, the MIC co-processors are also connected to the host CPU processors via PCIe. And the MIC also supports a lot of programming models such as OpenMP, MPI and etc.

To implement AWP-ODC on the MIC-based supercomputers, we can utilize the same parallel programming framework presented in this dissertation, because the hardware architectures are quite similar between MIC-based and CPU-GPU based supercomputers. Computation data also needs to be offloaded into the MIC co-processors and data communication between MIC co-processors still needs to bypass the slow PCIe connection. The main difference is the computation optimization on single computing node. The MIC co-processor provides more independent hardware threads and can handle more generation computation including if statements or data divergence. In addition, full use of the 512-bit SIMD is needed for vector computation to allow the maximum performance gain.

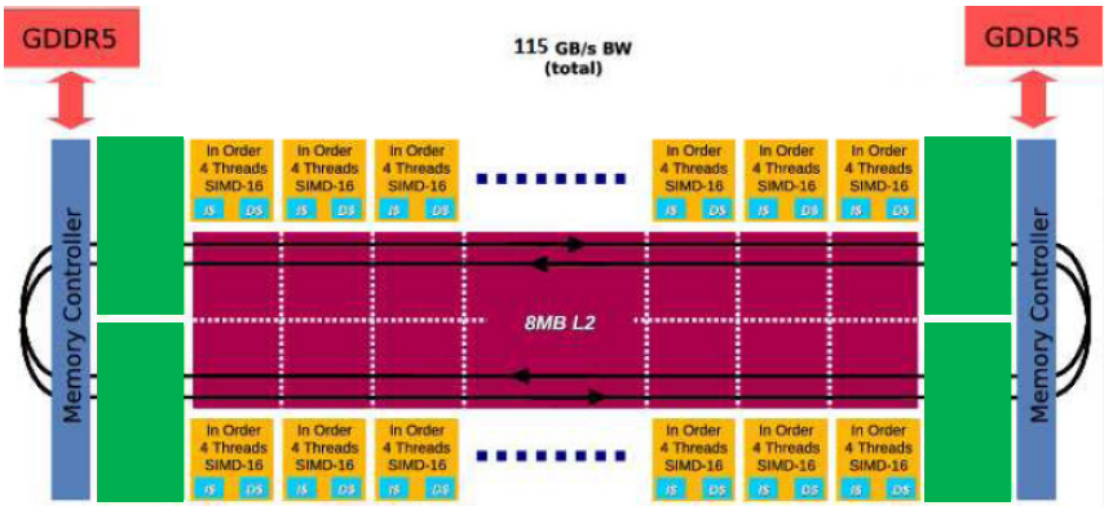


Figure 7.2: The architecture of Intel MIC co-processors: including 61 in-order cores and 4 hardware threads per core. Each core has private L1 cache (32KB I-cache and D-cache) and shared L2 cache (512KB unified per core) [88].

7.3 Dissertation Conclusions

A highly scalable finite-difference time-domain seismic simulation is presented in this dissertation on the world-class petascale heterogenesous supercomputer, targeting 3D earthquake hazard calculation with faster time-to-solution and higher computation efficiency. The primary goal of this work is the design, implementation, analysis and optimization of the well-known AWP-ODC application towards petascale computing on the largest CPU-GPU based supercomputer (Titan at ORNL). The production simulation using this new development has generated realistic dynamic earthquake source description and detailed physics-based anelastic ground motions at frequencies pertinent to scientific research or industry purpose (e.g. safe building design).

Heterogeneous computing has becoming more and more popular in HPC. The computing architecture has evolved to the traditional CPU plus accelerators instead of the CPU-only. Even the world leading CPU vendor Intel has introduced MIC architecture as a flagship product to compete for HPC market. Based on the Top 500 List released in Nov. 2013, only the No.1 supercomputer Tianhe-2 in China is installed with the MIC architecture, while the most popular heterogeneous supercomputers areremained with the hybrid CPU/GPU design. Our next step will be to migrate this GPU-based code into the Intel MIC, counting Intel's interest to continue the

support of the many-core architectures.

To enable petascale earthquake simulations on supercomputers, data management is very critical as many terabytes and millions of 3D input and output files are generated on petascale supercomputers. The end-to-end earthquake simulation workflow developed in this work help manage the earthquake simualtion with fault tolerance capability features added to secure the accuracy and correctness of input/output data. The protocol workflow framework developed for data transfer between HPC sites demonstrated a saving of more than 73% data transfer time. The job management scheme based on iRODS system for data pre-pocessing and archieval achieved almost 5x faster than conventional iRODs methods.

Although our scientifc workflow handles the data pre/post-processing, the kernel part of the petascale earthquake simulation is the numerical simulation solver. The optimization of the solver are two-folds: the first is the single-GPU optimization which focuses on improving data locality; and the second is the multi-GPU implementation which tends to miminize the frequency of data communication in each iteration, thus allowing fully overlap of data communication.

AWP-ODC as a memory-bound application (involving 21 3D variables kept on memory) with very low FLOPs/Bytes ratio (around 0.508), its computation performance is dominated by the arithmatic throughput. We completely re-wrote the AWP-ODC code using C/CUDA: first to put constant 3D variables into the GPU read-only memory to take advantage of read-only memory cache, then to design proper data decomposition strategy to make sure threads in the same wrap can access adjacent data for better cache hit rate. Memory padding guarantees each data fetching is from the same memory page and improves data fetching efficiency. Register optimization is to fully reuse the data already stored in registers and minimize the global memory access. L1 cache/shared memory optimization is to make full use of the fast on-chip memory. Our single GPU memory-bound code achieved remarkable 143.8 GFLOPs performance on NVIDIA M2090, which is approximately 10% of the peak performance.

For multi-GPU implementation, a notable communication model has been developed to hide the communication latency, especially to overlap the high latency caused by the data communication between CPU and GPU. Firstly, we did two-layer data decomposition for CPU and GPU to make sure communication only happens in y and z directions (fast memory directions), then we extend the computation region for stress to reduce data communication for stress component. In-order communication guarantees data in ghost cell regions are correct and

also miminize the frequency of data passing through the slow PCIe bus. The overlapping method has fully overlapped the data communication time by the computation time and helped to avoid degrading the computation performance achieved by single-GPU optimization. The benchmark experiements on Titan demonstrated an excellent weak scaling up to full machine and a sustained performance up to 2.3 PetaFLOPs, the highest earthquake simulation performance to date.

The social impact of our research work can be summarized in two recent real-world production earthquake simulations: the first a 0-10Hz deterministic earthquake simulation on the Titan system, and the second the 140-sites CyberShake 14.2 production calculations using the GPU-based code on the NCSA Blue Waters system. The 10-Hz run is slightly larger than the M8 simulation performed in 2010, with a simulation time of approximately 5.5 hours using 16K GPU nodes compared to previous 24 hours run using 220K CPU cores. As the first sustained PetaFLOP seismic production simulation by Jan. 2014, this is a new record for earthquake simulation in terms of scale and scalability. The new AWP-ODC code is expected to save more than 500 million computation core-hours for the proposed statewide CyberShake 1.0-Hz model.

Recognized by the research community [89], the social media [90] and the technology companies [91], and HPC User Forum [7], this work achieves we promised in terms of enabling petascale seismic production simulation with dramatically reduced time-to-solution and power consumption. In the end, the GPU-based AWP-ODC code is under development for a public release and tutorial by UCSD HPGeoC laboratory, with a goal of expanded use of the software by earthquake researchers and technical developers around the world.

Bibliography

[1] Top 500 Supercomputer List: http://www.top500.org/list/

[2] 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.

[3] Inside HPC: Four Gordon Bell Finalists using GPU-powered Titan Supercomputer, 2013. http://insidehpc.com/2013/10/31/four-gordon-bell-finalists-using-gpu-powered-titan-supercomputer/.

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

[5] 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.

[6] SCEC CyberShake Project: http://scec.usc.edu/research/cme/groups/cybershake.

[7] HPC Wire News released in June 24th, 2013 (Source from UC San Diego): SDSC Geocomputing Lab named winner of HPC Innovation Excellence Awarded by IDC http://archive.hpcwire.com/hpcwire/2013-06-24/sdsc_geocomputing_lab_named_winner_of_hpc_innovation_excellence_award_by_idc.html

[8] G. E. Moore, Cramming More Components onto Integrated Circuits. Proceedings of The IEEE, Vol. 86, No. 1, January 1998.

[9] T. Mudge, Power: A First-Class Architectural Design Constraint. IEEE Computer, Vol. 34, Issue: 4, Page 52-58. April, 2001.

[10] Kraken Supercomputer: http://www.nics.tennessee.edu/computing-resources/kraken

[11] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik and O. O. Storaasli. State-of-the-art in heterogeneous Computing. Scientific Programming, Vol. 18. Page 1-33. 2010.

[12] IBM RoadRunner Supercomputer: http://en.wikipedia.org/wiki/IBM_Roadrunner

[13] Top 500 Supercomputer News in June, 2008: International Supercomputing Conference To

Host First Panel Discussion on Breaking the PetaFlops/s Barrier. http://top500.org/blog/2008/06/09/international_supercomputing_conference_host_first_panel_discussion_breaking_petaflop_s_barrier

[14] Los Alamos National Laboratory: First Science at The Petascale – Results from the Roadrunner Supercomputer. Open Science, Oct, 2009.

[15] NVIDIA CUDA 1.0 Release: http://www.beyond3d.com/content/news/304

[16] General Purpose Graphics Processing Units : http://en.wikipedia.org/wiki/GPGPU

[17] TITAN Supercomputer: http://www.olcf.ornl.gov/computing-resources/titan/

[18] NVIDIA Cooperation. http://developer.nvidia.com/content/tesla-m2090-announced

[19] NVIDIA Cooperation. NVIDIA's Next Generation CUDA Compute Architecture, Kepler GK110. White Paper, 2012.

[20] Intel Cooperation. Intel Many Integrated Core (Intel MIC) Architecture. International Supercomputing (ISC'11), Demos and Performance Description, June, 2011.

[21] Intel Cooperation-Many Integrated Core (MIC). Knights Corner launches the technology. http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html

[22] IEEE Spectrum: China's Tianhe-2 Caps Top 10 Supercomputers. Jun 17, 2013. http://spectrum.ieee.org/tech-talk/computing/hardware/tianhe2-caps-top-10-supercomputers

[23] TACC Stampede Supercomputer: https://www.tacc.utexas.edu/.

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

[25] LINPACK, Wikipedia: http://en.wikipedia.org/wiki/LINPACK

[26] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation). The MIT Press, 2007. 18, 27, 29

[27] OpenMP: Wikipedia: http://en.wikipedia.org/wiki/OpenMP

[28] MPI: Wikipedia: http://en.wikipedia.org/wiki/Message_Passing_Interface

[29] NVIDIA News: NVIDIA Dramatically Simplifies Parallel Programming with CUDA 6. http://nvidianews.nvidia.com/Releases/NVIDIA-Dramatically-Simplifies-Parallel-Programming-With-CUDA-6-a62.aspx

[30] Khronos Group: http://www.khronos.org

[31] OpenCL, Wikipedia: http://en.wikipedia/wiki/OpenCL

[32] RenderScript: http://developer.android.com/guide/topic/renderscript/

[33] DirectCompute, Wikipedia: http://en.wikipedia.org/wiki/DirectCompute/

[34] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of CUDA and OpenCL. In The 40-th International Conference on Parallel Processing (ICPP'11), Taipei, Taiwan, September 2011.

[35] OpenACC: Directives for Accelerators: http://www.openacc-standard.org

[36] OpenACC, Wikipedia: http://en.wikipedia.org/wiki/OpenACC

[37] C. Yang, W. Xue, H. Fu, L. Gan, L. Li, Y. Xu, Y. Lu, J. Sun, G. Yang and W. Zheng. A Peta-scalable CPU-GPU algorithm for global atmospheric simulations. In PPoPP 13: proceedings of 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, February 23-27, 2013, Shenzhen, China, pp. 1-12.

[38] 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.

[39] 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

[40] 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.

[41] 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.

[42] 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.

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

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

[45] T. Okamoto, H. Takenaka, T. Nakamura, and T. Aoki. “Accelerating Simulation of Seismic Wave Propagation by Multi-GPUs”. AGU2010 Meeting, San Francisco, California, Dec. 13-17, 2010.

[46] 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.

[47] NSF TeraGrid Facility, http://www.globus.org/solutions/tgcp/

[48] Extreme Science and Engineering Discovery Environment (XSEDE), https://www.xsede.org/

[49] National Institute for Computational Sciences. Oak Ridge National Laboratory at University of Tennessee. http://www.nics.tennessee.edu/computing-resources/kraken, 2013

[50] Sourthern California Earthquake Center, http:// www.scec.org

[51] Y. Cui, A. Chourasia, R. Moore, K. Olsen, P. Maechling, T. Jordan, The TeraShake Computational Platform, Advances in Geocomputing, Lecture Notes in Earth Sciences 119, DOI 10.1007/978-3-540-85879-9_7, pp229-278, editor H. Xing, Springer-Verlag Berlin Heidelberg, 2009.

[52] K.B. Olsen, S.M. Day, J.B. Minster, Y. Cui, A. Chourasia, D. Okaya, P. Maechling and T.H. Jordan, “TeraShake2: spontaneous rupture simulations of Mw 7.7 earthquakes on the southern San Andreas fault”, Bull. Seism, Soc, Am. Vol.98, no.3, pp. 1162-1185, June 2008, doi:10.1785/0120070148.

[53] K. Lee, Y. Cui, T. Kaiser, P. Maechling, K.B. Olsen, and T.H. Jordan, “I/O Optimizations of SCEC AWP-Olsen Application forvPetascale Earthquake Computing,” Supercomputing conf. (SC’09), 2009, Poster.

[54] P. Maechling, E. Deelman, and Y. Cui, “Implementing Software Acceptance Tests as Scientific Workflows,” PDPTA CSREA Press, 2009, pp. 317-323.

[55] Amit Chourasia: M8 Quake. http://users.sdsc.edu/~amit/web/viz/m8

[56] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. Data Management and transfer in high-performance computational grid environments. Parallel Computing Journal, Vol. 28, pp.749-771. 2002.

[57] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel and S. Tuecke. Secure, efficient data transport and replica

management for high-performance data-intensive computing. IEEE Mass Storage Conference, San Diego, CA, Apr, 2001.

[58] R. Madduri, W. Allcock, C. Hood. Reliable File Transfer in Grid Environments. Proceedings of the 27th IEEE Conference on Local Computer Networks, pp737-738.

[59] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda and S. Sekiguchi. Grid DataFarm Architecture for Petascale Data Intensive Computing. Proceedsings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), pp.102-110, 2002.

[60] D. Thain, J. Basney, S. Son, M. Livny. The Kangaroo Approach to Data Movement on the Grid. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), 2001.

[61] IRODS: Data Grids, Digital Libraries, Persistent Archives, and Real-time Data Systems. https://www.iRODS.org/ , 2013

[62] I. Foster, Globus toolkit version 4: Software for service-oriented systems. Proceedings of IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005. http://www.globus.org/

[63] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, S. Chen, L. Gilbert, C. Hou, C. Lee, R. Marciano, P. Tooby, A. Torcy, B. Zhu. iRODS Primer: integrated Rule-Oriented Data System.(Book to be published)

[64] IA-64 Linux Cluster, San Diego Supercomputer Center, University of California, San Diego. http://www.sdsc.edu/us/resources/ia64/

[65] SAM (Storage Archive Manager)-QFS, San Diego Supercomputer Center (SDSC). http://www.sdsc.edu/us/resources/samqfs/index.html

[66] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. Request for Comments 1323, May 1992.

[67] Simone, A., and S. Hestholm, Instabilities in applying absorbing boundary conditions to high-order seismic modeling algorithms, Geophysics, 63,1017– 1023, 1998.

[68] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multi-core architectures. Commun. ACM, 52:65-76. April 2009.

[69] H. Nguyen, Y. Cui, K.B. Olsen, K. Lee, "Single CPU optimization of SCEC AWP-Olsen," SCEC Annual Meeting, 2009, Poster.

[70] P. N. Glaskowsky. NVIDIA's Fermi: The First Complete GPU Computing Architecture.

NVIDIA White Paper, 2009.

[71] NVIDIA C Programming Guide, Version 4.0. NVIDIA Cooperation. May, 2011.

[72] 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.

[73] A. Schafer, D. Fey. High Performance Stencil Code Algorithms for GPGPUs. In Proceeding of International Conference on Computational Science, pp.2027-2036. ICCS 2011.

[74] Accelerating a 3d finite-difference earthquake simulation with a c-to-cuda translator. D. Unat, J. Zhou, Y. Cui, S.B. Baden, X. Cai. Computing in Science & Engineering, 2012

[75] S. Potluri, P. Lai, K. Tomko, S. Sur, Y. Cui, M. Tatineni, K. W. Schulz, W. L. Barth, A. Majumdar and D. K. Panda. Quantifying Performance Benefits of Overlap using MPI-2 in a Seismic Modeling Application. International Conference on Supercomputing, Tsukuba, Ibaraki, Japan, 2010 (ICS'2010).

[76] Blue Water Supercomputer: http://www.ncsa.illinois.edu/BlueWaters/system.html

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

[78] Olsen, K. et al., Ground Motion Prediction From Low-velocity sediments including Statistical Models of inhomogeneities in Southern California Basins. http://scec.usc.edu/scecpedia/SDSC/NCAR_Viz.

[79] Z. Shi and S. M. Day, "Rupture dynamics and ground motion from 3-D rough-fault simulations," Journal of Geophysical research, vol. 118, pp. 1-20, 2013

[80] R. Taborda and J. Bielak, "Ground-motion simulation and validation of the 2008 Chino Hills," Bulletin of the Seismological Society of America, vol. 103, pp. 131-156, 2013.

[81] E. H. Field, T. E. Dawson, K. R. Felzer, A. D. Frankel, V. Gupta, T. H. Jordan, T. Parsons, M. D. Petersen, R. S. Stein, R. J. Weldon II, and C. J. Wills, "Uniform california earthquake rupture forecast, version 2 (UCERF 2)," Bulletin of the Seismological Society of America, vol. 99, no. 4, pp. 2053-2107, August 2009.

[82] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi, "CyberShake: A physics-based seismic hazard model for southern california," Pure and Applied Geophysics, vol. 168, no. 3, pp. 367-381, March 2011.

[83] S. Callaghan, E. Deelman, D. Gunter, G. Juve, P. Maechling, C. Brooks, K. Vahi, K. Milner,

R. Graves, E. Field, D. Okaya, and T. Jordan, "Scaling up workflow-based applications," Journal of Computer and System Sciences, vol. 76, no. 6, pp. 428–446, September 2010.

[84] P. Chen, L. Zhao, and T. H. Jordan, "Full 3D tomography for the crustal structure of the Los Angeles region," Bulletin of the Seismological Society of America, vol. 97, no. 4, pp. 1094-1120, 2007.

[85] P. Chen, T. H. Jordan, and L. Zhao, "Full three-dimensional tomography: a comparison between the scattering-integral and adjoint-wavefield methods," Geophysical Journal International, vol. 170, no. 1, pp. 175-181, 2007.

[86] C. Tape, Q. Liu, A. Maggi, and J. Tromp, "Seismic tomography of the southern California crust based on spectral-element and adjoint methods," Geophysical Journal International, vol. 180, no. 1, pp. 433-462, 2010.

[87] Y. Cui, E. Poyraz, J. Zhou, S. Callaghan, P. Maechling, P. Chen and T. H. Jordan, “Accelerating CyberShake Calculations on XE6/XK7 Platforms of Blue Waters”, Extreme Scaling Workshop 2013, August 15-16, Boulder, 2013.

[88] Intel Cooperation: Beginning Intel Xeon Phi Coprocessor Workshop Introduction, 2012. http://software.intel.com/sites/default/files/Beginning%20Intel%20Xeon%20Phi%20Coprocessor%20Workshop%20Introduction.pdf

[89] OLCF: Titan Simulates Earthquake Physics Necessary for Safer Building Design https://www.olcf.ornl.gov/2013/12/16/titan-simulates-earthquake-physics-necessary-for-safer-building-design/

[90] HPC Wire: Researchers Develop Code that Reduces Time and Cost in Simulating Seismic Hazards in Apr.02, 2013 from San Diego Supercomputer Center. http://archive.hpcwire.com/hpcwire/2013-04-02/researchers_develop_code_that_reduces_time_and_cost_in_simulating_seismic_hazards.html

[91] NVIDIA Cooperation - CUDA SPOTLIGHT: GPU-Accelerated Earthquake Simulations. http://www.nvidia.com/content/newsletters/web/CUDA-Week-in-Review-july-03-13-web.html