Accelerating CyberShake Calculations on the XE6/XK7 Platform of Blue Waters

Y. Cui, E. Poyraz and J. Zhou
San Diego Supercomputer Center
University of California, San Diego
La Jolla, USA
yfcui,epoyraz,j4zhou@ucsd.edu

S. Callaghan, P. Maechling and T.H. Jordan
Southern California Earthquake Center
University of Southern California
Los Angeles, USA
scottcal,maechlin,tjordan@usc.edu

L. Shih
Department of Computer Engineering
University of Houston - Clear Lake
Houston, TX USA
shih@uhcl.edu

P. Chen
Department of Geology and Geophysics
University of Wyoming
Wyoming, USA
pchen@uwyo.edu

***Abstract*— CyberShake is a computational platform developed by the Southern California Earthquake Center (SCEC) that explicitly incorporates earthquake rupture time histories and deterministic wave propagation effects into seismic hazard calculations through the use of 3D waveform simulations. Using CyberShake, SCEC has created the first physics-based probabilistic seismic hazard analysis (PSHA) models of the Los Angeles region from suites of simulations comprising ~10^8 seismograms. The current models are, however, limited to low seismic frequencies (≤ 0.5 Hz). To increase the maximum simulated frequency to above 1 Hz and produce a California state-wide model, we have transformed SCEC Anelastic Wave Propagation code (AWP-ODC) to include strain Green's tensor (SGT) calculations to accelerate CyberShake calculations. This tensor-valued wavefield code has both CPU and GPU components in place for flexibility on different architectures. We have demonstrated the performance and scalability of this solver optimized for the heterogeneous Blue Waters system at NCSA. The high performance of the wave propagation computation, coupled with CPU/GPU co-scheduling capabilities of our workflow-managed systems, make a statewide hazard model a goal reachable with existing supercomputers.**

Keywords—CyberShake; AWP-ODC; Strain Green Tensor; Co-scheduling; Topology; Scalability; Seismic Hazard Map

I. INTRODUCTION

The CyberShake project is a complex, integrative, high-risk, high-reward computational research activity coordinated by the Southern California Earthquake Center, supported by the USGS and NSF, that requires advancements across both geoscientific and computing domains [17]. CyberShake utilizes 3D simulations and finite-fault rupture descriptions to compute deterministic (scenario-based) and probabilistic seismic hazard in Southern California [12]. Long period effects such as coupling of directivity and basin response that cannot be captured with standard approaches are clearly evident in CyberShake hazard maps. Moreover, CyberShake allows for rapid recomputation of the hazard map to reflect short-term probability variations provided by operational earthquake forecasting. Going beyond traditional hazard analysis, event-specific phenomena can also be identified and analyzed through examination of the individual ground motion waveforms. This process highlights the importance of key elements in the Earthquake Rupture Forecast (ERF) that are required by the simulation approach, including magnitude-rupture area scaling, aleatory and epistemic magnitude variability and spatio-temporal rupture characterization (Figure 1).

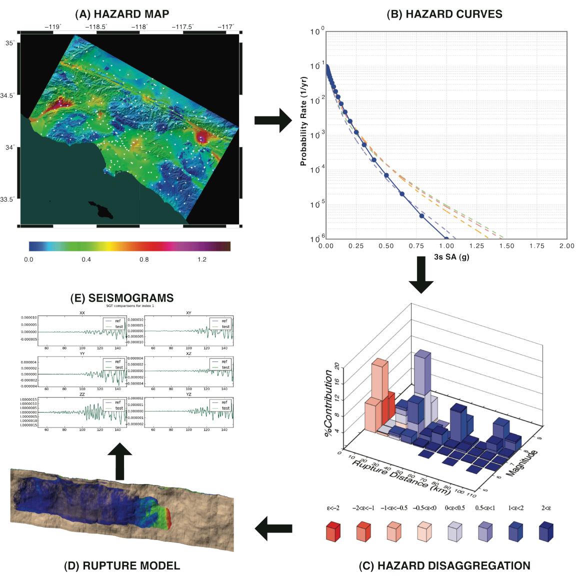


Figure 1: The CyberShake hazard model, showing the layering of information. (A) Hazard map for the LA region (hot colors are high hazard). (B) Hazard curves for USC site. (C) Disaggregation of hazard in terms of magnitude and distance. (D) Rupture with the highest hazard at the site. (E) Seismograms 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).

SCEC researchers have identified Los Angeles and San Francisco as the geographical regions that are the top scientific and computational priorities for further CyberShake hazard calculations. Once these urban seismic hazard studies have been completed, we will apply the technique to other, less populated regions in California.

The essential elements that must come together for an accurate CyberShake-1Hz hazard calculation include the following: (1) 3D velocity model for California; (2) UCERF2.0 extended ERF rupture descriptions; (3) computational engine for higher frequency (>1Hz) wave propagation simulations; (4) ensemble capabilities at large-scale to post-process the wave propagation simulations. However, the key computational engine associated with this procedure is a tensor-valued wavefield calculation that requires parallel computing of ground shaking calculations at high frequency. A California state-wide seismic hazard map at the maximum frequency of 1 Hz requires intensive ground shaking calculations for a few thousand sites. Thus, to advance and apply earthquake system science research within the practical limits of currently available HPC resources, we must aggressively improve the computational performance of our physics-based ground motion simulation software.

In this paper, we will introduce our recent development of a CyberShake platform for use on heterogeneous CPU/GPU systems, that includes an anelastic wave propagation code for tensor-valued wavefield calculations, and a co-scheduling runtime environment for post-processing of seismograms calculations. Section II of this paper introduces the standard AWP-ODC software and the numerical methods for reciprocity calculations. Section III verifies the correctness of Strain Green's Tensor (SGT) creation, seismogram synthesis, and error minimization from numerical differentiation, and applies these new capabilities to obtain the first hazard curve on Blue Waters. Section IV summarizes information about Blue Waters systems. Section V introduces the parallel efficiency and the scalability achieved on petascale supercomputers. Section VI introduces the topology-aware performance tuning. Section VII discusses the co-scheduling enabled on Blue Waters for effective use of a hybrid system. We conclude the paper with outlooks of the project short-term goals.

II. Strain Green Tensor Calculations With AWP-ODC

The Anelastic Wave Propagation software AWP-ODC (hereafter abbreviated AWP, AWPc for the CPU code, and AWPg for the GPU code) is a fourth order finite difference modeling code that solves 3D velocity-stress wave equations with the explicit staggered-grid scheme. This code, capable of both dynamic rupture and earthquake wave propagation simulations, is a community code used by SCEC for large-scale ground motion simulations [5, 15, 20, 21].

We discuss here the implementation of a tensor-valued wavefield calculation based on AWP that can produce ground motions from many single-site ruptures efficiently.

A. Algorithmic Backgrounds of SGT Calculations

In seismology, the strain tensor is often used to describe the deformation of the earth medium caused by seismic-wave-generated displacement field and is linearly related to the stress field by the constitutive law [19]. We call the strain field generated by a unit impulsive force located at $\boldsymbol{r}_s$ and pointing in the $\hat{x}_n$ direction the SGT, i.e.,

$$H_{ijn}(\boldsymbol{r},t;\boldsymbol{r}_s) = \frac{1}{2}[\partial_i^S G_{jn}(\boldsymbol{r},t;\boldsymbol{r}_s) + \partial_j^S G_{in}(\boldsymbol{r},t;\boldsymbol{r}_s)] \quad (1)$$

where $\boldsymbol{r}$ is the receiver location and G_{in} is the ith component of the displacement response to the nth component of a point force at $\boldsymbol{r}_S$, and the spatial gradient operator acts on the field coordinate $\boldsymbol{r}$ [15]. The reciprocity of the Green's tensor, i.e., $H_{ijn}(\boldsymbol{r},t;\boldsymbol{r}_s) = H_{jin}(\boldsymbol{r}_s,t;\boldsymbol{r})$ enables us to express the synthetic displacement seismogram generated by an arbitrary source and recorded at a fixed receiver location using the SGT computed for the receiver location, i.e.,

$$u_n(\boldsymbol{r},t;\boldsymbol{r}_s) = \boldsymbol{H}(\boldsymbol{r}_S,t;\boldsymbol{r}){:}\,\boldsymbol{M} \quad (2)$$

where $\boldsymbol{M}$ is the seismic moment tensor. The SGT can be computed from the stress-field, which is explicitly computed in our AWP code, by applying the stress-strain constitutive relation.

The elements of the SGT can be used in earthquake source parameter inversions to obtain the partial derivatives of the seismograms with respect to the moment tensor elements. By directly using the strain Green tensor, we can improve the computational efficiency in waveform modeling while eliminating the possible errors from numerical differentiation [3, 4, 15]. Seismic reciprocity can then be applied to compute synthetic seismograms from SGTs, from which peak spectral acceleration values are computed and combined into hazard curves [12].

B. Implementation of CPU-based SGT creation

We start with implementing SGT in CPU-based AWP (hereafter abbreviated AWP-SGTc). In this version, absorbing boundary conditions include both the split-equation Perfectly Matched Layers and 'sponge layers' Cerjan [2]. The SGT implementation is adapted from co-author Po Chen's schemes that were originally used to compute the Green's tensors for point impulsive body-forces located at the receiver locations for full-waveform tomography calculations.

AWP-SGTc decomposes the simulation domain in 3D. For each subgrid of the simulation domain velocity and stress are computed by the responsible processor. This code exchanges ghost cell data twice per subgrid, for velocity and stress with six neighbors, at each iteration of the computation loop.

Strain tensor inputs are different from standard AWP. The input parameter *igreen* specifies how the input file should be processed. There are eight different input modes that can be chosen. First is the standard AWP source input. In this mode stress tensors at the source points are given as input. Four of the input modes set initial velocities at the source points for *X*, *Y*, or *Z* directions, or all three directions. The remaining three modes set initial stress tensors on *XZ*, *YZ*, and *ZZ* faces at the source points. These input modes allow flexibility for the use of various inputs in the simulation.

In a separate input file, the solver reads in a list of receivers for which the strain tensor variables are saved at specified time step instances. Since the number of receivers is typically smaller than the number of grid points by a factor of 1,000, the solver reads in this file serially. Then the master CPU distributes the receivers to their respective CPUs.

The output of AWP-SGTc is also different from AWPc. Each CPU core saves the six strain tensor variables for the receivers which they handle at the specified time step instances for a specified number of times. Then using collective MPI-IO, each core writes out its tensor results to one large, appropriately striped (in Lustre) file.

The memory requirement of AWP-SGTc is larger than AWPc. In the initialization step, application computes two SGT constants per grid point that are used. Then, these two 3D arrays have to reside in the memory throughout the simulation. Compared to AWPc, this causes approximately a 10% increase in the memory requirement.

C. *Implementation of GPU-based SGT creation*

Adapting the solver using GPU is the need to accelerate the SGT generation, which accounts for approximately 90% of the CyberShake core-hours. This CUDA/MPI code (hereafter abbreviated AWP-SGTg), supports 2D decomposition on CPUs. Then each GPU on XK7 node GPU computes velocity and stress for its own subgrid of the simulation volume.

A CUDA kernel has been developed in the GPU to calculate the six tensor variables for the receivers that it handles, at the specified time step instances. The computation workload of these variables is distributed among different CUDA threads for each receiver on GPU, and thus is computed in parallel. Depending on the number of the receivers a GPU handles, this new kernel may speed up SGT calculation by a factor of 30,000 compared to the CPU code on XK7 system.

After the computation, we copy the results back to CPUs. Ideally we like to aggregate the results over time step instances on the GPUs, however, in practice there is a limitation because of the GPU memory constraint. We chose to use GPU memory for the computationally required data only. The copied SGT time series are aggregated on the CPU memory, before they are written out using collective MPI-IO [6].

The input modes, output format, and IO control parameters are the same as used in the CPU code. This allows inputs and outputs to be processed independently of the code used. Hence users have the flexibility in choosing the right code for the systems they use without worrying about the inputs and outputs, and future changes and improvements to IO can be quickly integrated into both codes.

The memory requirement of the AWP-SGTg is larger than the standard AWPc code. The application needs to keep two tensor constants and receiver coordinates in the GPU memory. This limits the maximum subgrid size that can be computed per GPU. In return we have a larger memory available on the CPU side per subgrid (XK7 node). That allows efficient aggregation of output data before flushing, and hence improves IO efficiency.

D. *Implementation of IO for SGT calculation*

AWPc supports multiple modes of serial and parallel IO schemes for calculating SGTs. Depending on the inputs and simulation settings, users can choose the most appropriate one.

AWP-SGT is capable of reading in a large number of dynamic sources and petabytes of heterogeneous velocity mesh inputs [6]. This code handles extended sources in a two-step approach, allowing reading temporal and spatial partitions simultaneously, with a capability of handling millions of kinematic sources converted from a dynamic rupture simulation. However, copying the source data to GPUs through PCIe is an additional challenge at runtime for the GPU code.

We support three different modes for reading the sources and the mesh: serial reading of a single file, concurrent reading of pre-partitioned files, and concurrent reading through MPI-IO. Source partitioning involves both spatial and temporal locality required to fit in the GPU memory. Parameters are introduced to control how often the partitioned source is copied from CPUs to GPUs. This feature allows CPUs to read in large chunks of source data to avoid frequent access to file system, thanks to the large memory available on XK7 node, while GPU only copies over the amount it can afford. Our implementation scales well the terabytes initial dataset.

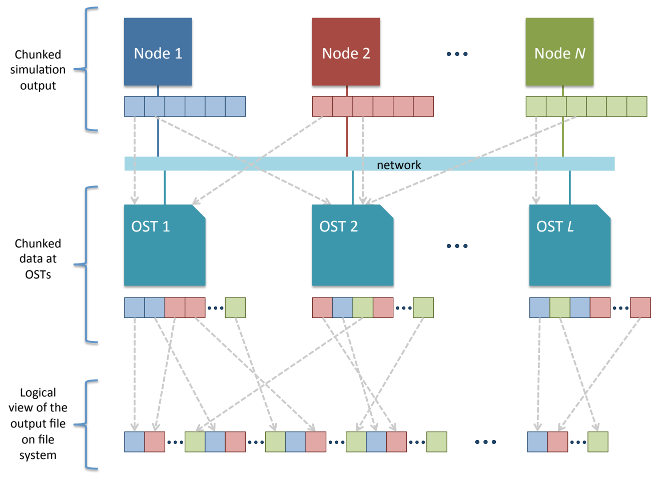

Chunked simulation output
Node 1
Node 2
Node N
network
Chunked data at OSTs
OST 1
OST 2
OST L
Logical view of the output file on file system

Figure 2: IO concept of AWP using MPI-IO. Each node computes and aggregates data to output. This results in logical chunks. At the time of writing out, these chunks are sent to object storage targets (OST) in parallel. IO communication costs are distributed to different portions of the network. OSTs manage the saved chunks. In the bottom of the figure, the logical view of the file represents interleaved chunks of data produced by different nodes.

AWP-SGT uses MPI-IO to write the simulation outputs to a single file concurrently. This works particularly well as more memory is available on CPUs to allow effective aggregation of outputs in CPU memory buffers before being flushed. This concept is depicted in Figure 2. When the specified number of aggregation is reached, each node communicates to a subset of available OSTs. This way a larger fraction of the network can be used to send output data to the filesystem, thus using available total bandwidth more efficiently. We also support run-time parameters to select a subset of the data points and timesteps to save.

On the Lustre parallel file systems, as used on NCSA Blue Waters, users are required to define "striping" for optimization purpose, by indicating how files within a specified directory are distributed across the filesystem. Files are broken into chunks, known as "objects," and stored to some number of "Object Storage Targets" (OSTs). Both the size of these objects and how many OSTs are used may be specified by the user. In the ideal situation, if all the CPU writers write the same amount of data (if the number of receivers are the same for each CPU), then the stripe size can be set to a size that evenly divides into the amount of data to be written per CPU. The stripe count (the maximum number of OSTs to be used) can be set to the number of writers up to the maximum number of OSTs supported by the filesystem. Then each node communicates to only one OST to write to the large, chunked file. That way the network is used in parallel and a balanced way. However if the amount of data to be written per CPU is too small or too large, it may be inefficient to set stripe size in this way [11].

For the CyberShake post-processing jobs, we use the *lfs setstripe* command to set stripe count as 10 and stripe size as 5 MB, to distribute the given file contents across the available OSTs. We observed 6.5X speedup compared to the default striping (stripe count 1 and stripe size 1 MB). Note that at the end of post-processing jobs, we place multiple SGT extractions with varying file sizes and writers depending on the jobs. Although not straightforward with striping options, the optimal parameters provide an overall superior IO rate.

III. VERIFICTION OF SGT IMPLEMENTATION

The AWP-SGT implementations have been extensively verified by comparing stress and strain outputs of earthquake sources to those from existing CPU codes. The AWP-SGTc code verification exercises, led by SCEC researchers K. Olsen, R. Graves, and others, is beyond the scope of this paper. We focus here on the GPU code verification compared to the verified and validated AWP-SGTc. We performed a variety of tests to ensure that AWP-SGTg produces results comparable in accuracy to the currently used SCEC CyberShake codes running on HPC systems.

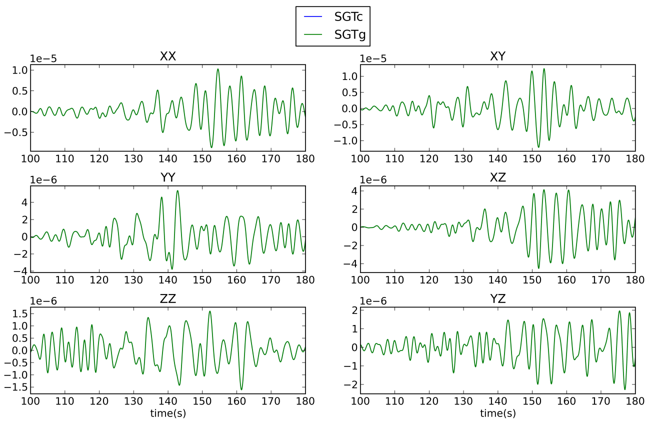


Figure 3: Comparison of SGTs generated using CPU and GPU versions of AWP-SGT. The test case site is USC.

A. *Verification of SGT Implementation Against Tensors*

We examined tensors generated using both codes. The results from the GPU code and reference model are nearly identical, with an average difference of 0.005%, in a 1.2 billion mesh point volume for 20K timesteps (Figure 3).

B. *Verification of SGT Implementation Against Hazard Curve*

PSHA results are typically delivered as seismic hazard maps, or as site-specific seismic hazard curves. Probabilistic seismic hazard curves relate peak ground motion on the *X*-axis to the probability of exceeding that level of ground motion on the *Y*-axis, for a site of interest. To verify AWP-SGTg, we calculated a hazard curve and compared it to one from AWP-SGTc. Figure 4 illustrates that the two versions produce very similar results (average difference 0.006%). Calculation of a hazard curve involves tensor time series data from over half a million locations in the volume, providing rigorous verification.

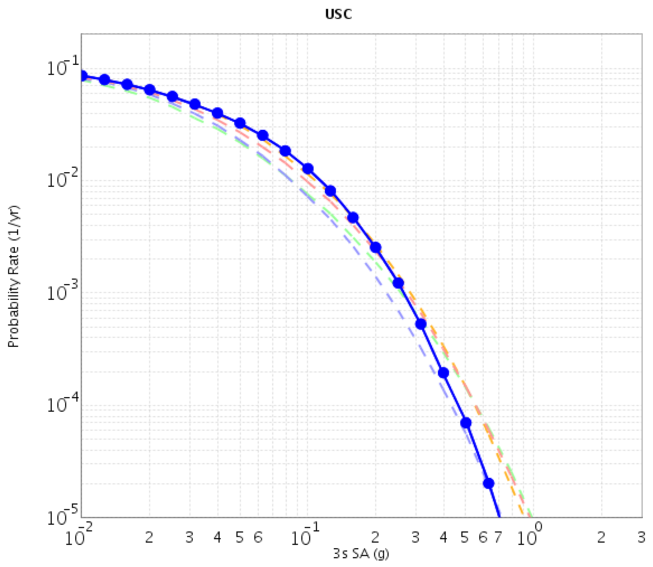


Figure 4: PSHA hazard curve calculated for the University of Southern California (USC) site. The horizontal axis represents ground motion at 3 seconds spectral acceleration, in terms of g (unit compared to the acceleration due to gravity). The vertical axis gives the probability of exceeding that level of ground motion. The blue line is the curve calculated using CyberShake with AWP-SGTg and co-scheduling, showing almost identical results compared to AWP-SGTc (difference ~0.006%). The dashed lines are hazard curves calculated using four common attenuation relationships which provide validation of the CyberShake methodology.

IV. BLUE WATERS SYSTEM

The Blue Waters system (hereafter abbreviated BW) of NCSA is a Cray hybrid machine composed of 237 cabinets of Cray XE6, AMD 6276 "Interlagos" Opteron processors with nominal clock speed of at least 2.3 GHz, plus 32 cabinets of Cray XK7 with NVIDIA K20X Tesla® Kepler™ accelerators [13]. The interconnect is Cray Gemini [18]. Two other systems are added for comparison purposes: Titan and Keeneland. The OLCF Titan is a Cray XK7 supercomputer located at the Oak Ridge Leadership Computing Facility (OLCF), with a theoretical peak double-precision floating point performance of more than 20 Petaflops. Titan 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 GPU, and 32 GB of RAM. Two nodes share a Gemini™ high-speed interconnect router. Nodes within the compute partition are connected in a 3D torus [13]. The Keeneland Full Scale (KFS) system consists of a 264-node cluster based on HP SL250 servers. Each node has 32 GB of host memory, two Intel Sandy Bridge CPU's, three NVIDIA M2090 (Fermi) GPUs, and a Mellanox FDR InfiniBand interconnect. The total peak double precision performance is around 615 TFlops [11].

V. SCALABILITY AND PERFORMANCE OF AWP-SGT

The strong scaling benchmarks are presented in Figure 5, the various test sizes are due to the difference of the available on XE6 and XK7. The smallest XK7 case (with 206 million mesh points) and all XE6 tests are run on NCSA BW, whereas other XK7 tests were on OLCF Titan. While the CPU code demonstrates good scaling, the degradation in strong scaling is observed for the XK7 case. This is primary due to the increase in the ratio of total volume of the required halo region (which surrounds the simulation volume) to the subgrid volume as the number of GPUs used increases. Thus computation to communication ratio decreases, and overlapping of communication with computation becomes less effective. The strong scaling performance in XE6 case is better. We also have experimented with topology aware resource management for AWP as explained in section VI.

Figure 6 presents AWP's weak scaling performance on XSEDE Keeneland Initial Delivery System (KIDS), OLCF Titan and NCSA BW. We observed perfect (100%) linear speedup on KIDS up to 90 nodes, and on Titan up to 8192 nodes. On KIDS, AWPg achieved 10% of the peak performance. In the weak scaling tests each GPU computed a subgrid of 52 million mesh points.

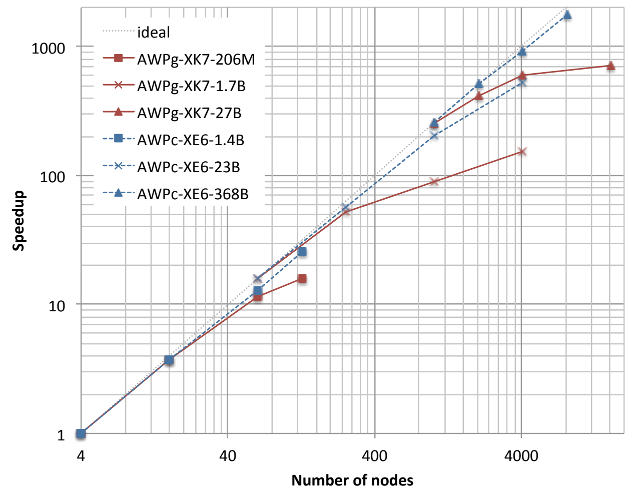


Figure 5: Strong scaling of AWP-SGTg and AWP-SGTc on NCSA Blue Waters (XE6/XK7) and OLCF Titan (XK7) with various problem sizes from 206M to 368B grid points.

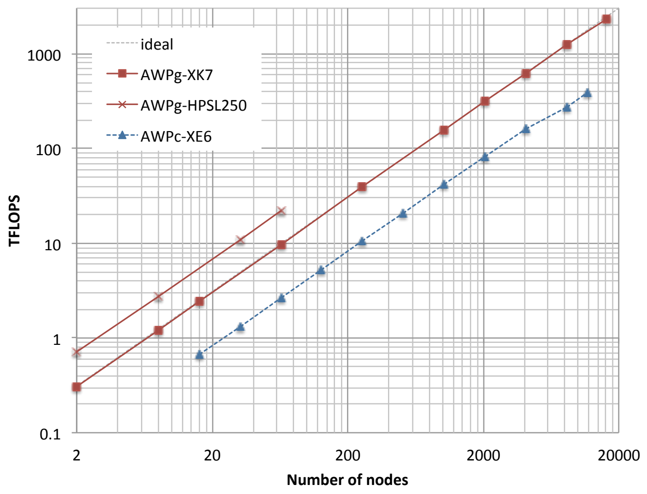


Figure 6: Weak scaling of AWP-SGTc and AWP-SGTg on NCSA Blue Waters (XE6/XK7) and OLCF Titan (XK7). Best performance was achieved on Keeneland (HPSL250, 3 GPUs per node used). Sustained 2.3 Petaflop/s was achieved on Titan using 16,640 XK7 nodes.

AWP-SGTg running on XK7 demonstrates a performance improvement of a factor of 3.7 compared to the AWP-SGTc running on XE6 [6]. Based on this information plus 5000 sites required to generate a California state-wide seismic hazard map with a maximum frequency of 1 Hz, our accelerated code will save more than 500 million allocation hours over the optimized AWP-SGTc software.

VI. TOPOLOGY PERFORMANCE TUNING

Overall, both the improved AWP-SGTc and AWP-SGTg achieved perfect weak scaling computation efficiency with the total computation workload scaled-up proportionally enough to feed the increased number of compute nodes used on petascale systems tested, as shown in Figure 6. As expected, strong scaling of those benchmarks tested starts to degrade as the number of computation nodes increases under fixed total computation workloads performed on NCSA BW, as depicted in Figure 5. Rather than a topology mismatch issue, the most likely culprit of strong scaling degradation of AWP-SGTg is the workload starvation of increased numbers of highly efficient number-crunching GPUs competing for the fixed total workload in strong scaling, plus the increase in halo region communication to computation proportion, as described in section V. But for AWP-SGTc, there may still be room for strong scaling improvement by optimizing the parallel resource mapping between virtual AWP-SGTc mesh topology and the allocated physical BW subnet topologies to reduce the communication hops.

With the help of R. Fiedler of Cray as well as G. Bauer and O. Padron of NCSA, we were able to visualize and inspect the network subnet fragmentation in the initial batch task placements on the default BW torus node allocation. As illustrated in Figure 7a, the 4,096 default physical XE6 compute nodes allocated by the BW batch job management system include distant communications spanning disjoint fragmented subnets shown in yellow, hopping through the red

XK7 (GPU capable) region. Figure 7b shows a more continuous 4096-node default subnet allocation during a lighter BW system load in blue vertical slab. Although the allocation is not as fragmented as the yellow subnet of Figure 7a, the flat slab allocation is stretching along the slower BW Y axis. The visualization of these less-optimal default network placement fragmentation presented us with a promising topology tuning opportunity to improve strong scaling, by allocating a continuous and more compact, cuboidal subnet within BW torus in order to better match AWP-SGTc's virtual 3D elongated near-neighbor mesh prism topology. Our topology-mapping optimization was designed to balance the tradeoffs among:

(1) Matching the virtual 3D Cartesian mesh topology of the typical 8x4x1 AWP-SGTc mesh proportion (e.g., 8960x4480x1120 for 45B mesh points) to an elongated physical subnet prism shape in the BW torus.

(2) Maximizing faster connected BW *XZ* plane allocation with the longest virtual mesh topology edge-aligned to the fastest communicating BW torus *Z* direction, producing a flatter subnet allocation and/or applying virtual mesh sheet folding to stretch over BW *XZ* planes. Note that subnet allocation extending to the edge of torus can also cut subnet diameter significantly due to wrap-around links.

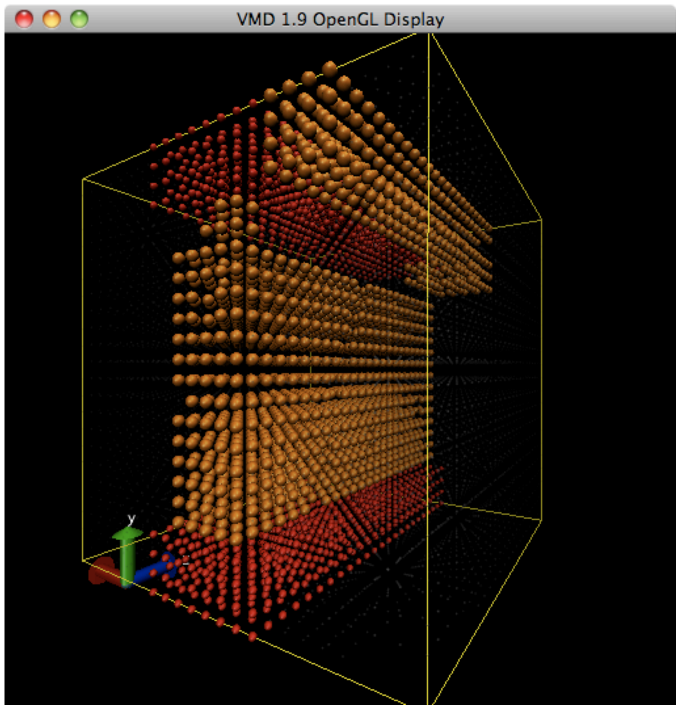
(3) Obtaining a tighter, more compact and cuboidal shaped BW subnet allocation for lower network diameter (distance between the most far-apart pair) to achieve efficient global barrier synchronization in the AWP-SGTc code.

(4) Reducing inter-node hops along the slowest BW torus *Y* direction, by increasing the partition grain size in BW *Y* direction, or by mapping BW *Y* axis to the shallowest virtual AWP V_Z axis, corresponding to the ground depth in the CyberShake hazard map.

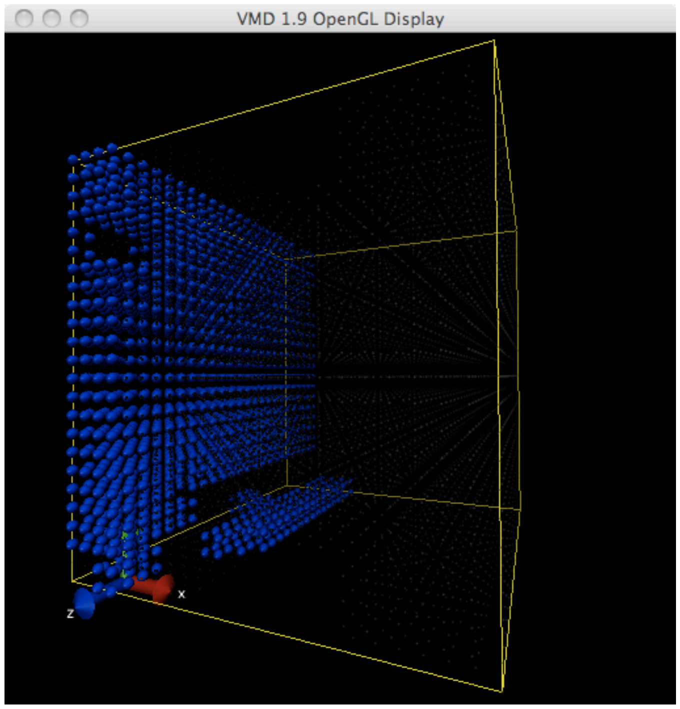
Ideally, we would optimize the virtual to physical network topology mapping by carefully reserving a best matching prism subnet in BW torus. However, BW's randomly situated down compute nodes at the time of run as well as non-computing IO service nodes make it difficult to accommodate every direct near-neighbor data exchange without inter-node communication hops. We thus employed Cray's Topaware tool to aid node selection and MPI rank (virtual process ID) ordering to harvest the potential speedup, similar to the reported improvement of 3% to 370% tested with other 2D/3D/4D Cartesian mesh applications [7, 8].

Given a requested logical subnet dimension, Topaware script explores the BW torus network topology to locate a slightly larger subnet prism skipping over randomly scattered down compute nodes and non-computing IO service nodes. For a strong scaling test run of 45 billion mesh grid points in a rectangular near-neighbor virtual topology of $8\times4\times1$ proportion in V_X, V_Y and V_Z axes, we tested Topaware's auto node selection and MPI rank ordering to further optimize 64-node, 512-node and 4096-node test cases.

Topaware tool is run by the following command, after setting some environment variables.



(a)



(b)

Figure 7: Default BW batch task placements on torus topology node allocation for AWP-SGTc on 4,096 XE6 nodes showing disjoint subnet fragmentations in yellow for (a) and blue for (b) scattered along the slowest network links in vertical Y dimension, hopping through the red XK7 region, where pin holes indicating possible IO nodes. (Visualization courtesy of NCSA BW staff O. Padron and G. Bauer).

```
% pick_nodes.sh NX NY NZ NVx NVy NVz S T M
```

where N_X, N_Y, and N_Z set the dimensions of the physical cuboid in 3D BW torus in terms of node pairs, N_{Vx}, N_{Vy}, N_{Vz} are the dimensions of the application's virtual prism per node pair, S (1, 2, or 3) sets the V_X, V_Y, V_Z axis of the virtual topology that is shared between each node in a node pair, T is the node type (32 cores for the two Opterons in each XE6 and 16 cores for the Opteron in each XK7 node), and M is the mode for which kind of nodes (e.g., down nodes) should be skipped in the search.

The environment variable *PN_MAP* sets the mapping of the virtual axes to physical ones in the torus. We set *PN_MAP* to 312 to map virtual V_X, V_Y, V_Z axes to physical BW torus *Z, X, Y* axes, respectively. After setting N_X, N_Y, N_Z, N_{Vx}, N_{Vy}, N_{Vz} and S according to the test, we set T=32 to use XE6 nodes for AWP-SGTc and M=0 to skip down compute nodes and IO service nodes.

A. 64-node Topaware-assisted AWP-SGTc strong scaling

To match the 3D virtual near-neighbor mesh topology of 8x4x1 proportion for the 64-node case, a block of $4\times1\times8$ Gemini routers, each with a node pair attached, was requested via *Topaware* script to obtain sufficient logical prism subnet skipping over some non-computing IO service or down nodes. The 64-node case was run with *4 1 8 4 4 4 3* setting, indicating $4\times1\times8$ torus node pairs requested, each node pair with $4\times4\times4$ MPI ranks in AWP-SGTc virtual topology, and the 3rd dimension is split between each node pair. Hence each 32-core XE6 node will have $4\times4\times2$ = 32 MPI ranks (Figure 8). The virtual V_X, V_Y, V_Z coordinates are mapped to *Z, X, Y* directions of the BW torus. Very slight speedup of 0.37% was achieved by the *Topaware* node selection over the already quite compact and optimal default task placement from 4.006 sec/step reduced to 3.991 sec/step for the 64-node test case.

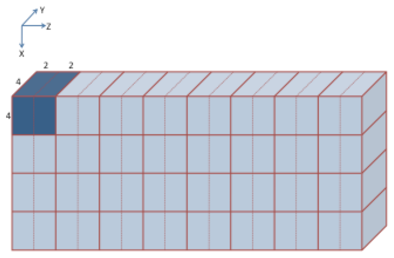


Figure 8: Representation of 64-node Topaware-assisted physical node allocation. The node list provided by Topaware tool has *4×1×8* node pairs. Each node pair maps to *4×4×4* MPI ranks. Solid red lines show the boundary between node pairs. Each node in a node pair shares the virtual topology in Z dimension, hence maintaining *4×4×2* virtual mesh topology. Dashed lines show the boundary between individual nodes in a node pair. The numbers on the lines represent how many MPI ranks are maintained in that direction in that section.

B. 512-node Topaware-assisted AWP-SGTc strong scaling

For the 512-node test run, Topaware helped selecting an 8x4x8 logical prism subnet block of Gemini node-pairs skipping over IO/down nodes. Topaware tool was run with *8 4 8 4 2 8 3* setting, with $8\times4\times8$ torus node pairs requested, loading each node pair with $4\times2\times8$ MPI ranks, splitting in 3rd dimension within each node-pair. We then have $4\times2\times4$ = 32 MPI ranks inside each node equipped with 32 cores. The speedup performance obtained for the 512-node benchmark is 3.15%, from 0.572sec/step reduced to 0.554 sec/step, showing a bit more promising than the 64-node case. The strong scaling efficiency is improved from 87.5% to 90% for 512 nodes. Among the benchmark cases tested, the more cuboid $8\times4\times8$ subnet performed better than the flatter subnets of $8\times2\times16$, which also outperforms the flattest *16×1×16 subnet allocation.*

C. 4096-node Topaware-assisted AWP-SGTc strong scaling

Similarly for testing the 4096-node case, Topaware script was launched to select a block of Gemini node-pairs of 16x8x16 logical prism subnet block skipping over service/down nodes. Topaware tool was run with *16 8 16 4 2 8 3* setting, meaning $16\times8\times16$ torus node pairs requested, with each node pair with $4\times2\times8$ MPI ranks, splitting in 3rd dimension within each node-pair. We then have $4\times2\times4$ = 32 MPI ranks inside each node equipped with 32 cores. The resulting continuous prism subnet allocation on BW torus is near optimal as shown in yellow in Figure 9, where the 2 cubes actually wrap around to connect directly as one continuous cuboidal prism. Topology tuning with Topaware tool achieved a significant 35.15% speedup, from 0.119 sec/step reduced to 0.077 sec/step, boosting the strong scaling efficiency from 52.6% to 81%.

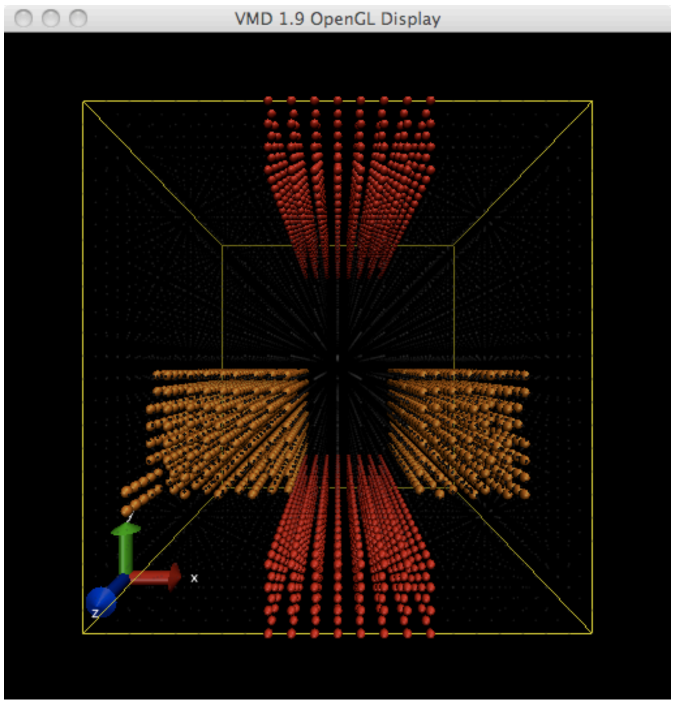


Figure 9: Topaware-assisted task placement on BW torus topology node allocation for AWP-SGTc on 4,096 XE6 nodes showing continuous subnet in yellow slab along the fastest XZ plane, imagining the left most face wrap-around touching directly to the right-most face. (Visualization courtesy of NCSA BW staff O. Padron and G. Bauer).

Table 1 summarizes the speedup of topology tuning up from 0.37% to 35% and improved strong scaling of fixed work load of 45 billion mesh points of 8960×4480×1120. Among all the benchmark tested for topology tuning of 64, 512, and 4096 nodes, the more compact and cuboidal prism subnets are better choices than the flatter subnet prims, because this particular chosen problem size dimension cannot be evenly factored to cover the whole BW XZ planes to take advantages of torus wrap-around connection.

Table 1: Topology tuning with Topaware tool improved strong scaling efficiency for fixed 45B mesh point AWP-SGTc benchmark calculation with 64, 512, and 4096 nodes

#nodes	Default	Topaware	Speedup	Efficiency ↑
64	4.006	3.991	0.37%	100% → 100%
512	0.572	0.554	3.15%	87.5% → 90%
4096	0.119	0.077	35.29%	52.6% → 81%

Topology mapping with automated Topaware assistance is easy to achieve for near optimal processor mapping, but challenging to fine tune further, due to BW's irregularly placed IO nodes. A compute node list file is available for looking up the corresponding Gemini network coordinates, but there is hardly a standard formula to translate between Gemini router network coordinates and compute node IDs. A benchmark test run can get struck in the job queue waiting without warning, if a selected node is down at the time. The job execution also encountered issues of long node-list handling when attempting to run 8K or more nodes. For AWP topology-aware scaling beyond 4,096 nodes, it will require further investigation beyond the help of Topaware.

VII. CPUs/GPUs Co-scheduling

When the SGT calculations are performed on GPUs, the CPUs on the same nodes are mostly idle except for handling IO and communications, a potential waste of resources. To maximize our results using the heterogeneous BW system, we have developed a runtime environment for co-scheduling across CPUs and GPUs. A CyberShake workflow consists of two phases: (1) two parallel AWP-SGT calculations, and (2) high-throughput reciprocity calculations with each rupture variation to produce seismograms and intensity measures of interest. Co-scheduling enables us to perform both phases of a CyberShake calculation simultaneously on XK7 nodes, reducing time-to-solution and making efficient use of available computational resources.

A. Multiple Innocation Co-scheduling

One approach to co-scheduling on XK7 nodes is to script the launching of multiple simultaneous parallel jobs. To run the CyberShake workflow, we use the following approach, outlined in pseudocode:

```
aprun -n 50  <GPU executable> <arguments> &
get the PID of the GPU job
cybershake_coscheduling.py:
    build all the cybershake input files
    divide up the nodes and work among a customizable number of jobs
    for each job:
        fork extract_sgt.py cores --> performs pre-processing and
        launches
        "aprun -n <cores per job> -N 15 -r 1 <cpu executable A>&"
        get PID of the CPU job
    while executable A jobs are running:
        check PIDs to see if job has completed
        if completed:launch
            "aprun -n <cores per job> -N 15 -r 1 <cpu executable B>&"
    while executable B jobss are running:
        check for completion
check for GPU job completion
```

To enable co-scheduling, we launch multiple MPI jobs on XK7 nodes via multiple calls to *aprun*, the ALPS utility to launch jobs on compute nodes from a Node Manager (MOM) node. We use core specialization, an *aprun* command line option, when launching the child *aprun* calls to keep one core available for GPU data transfer and communication calls, as both the GPU and CPU codes use MPI. Testing has shown that this co-scheduling approach results in minimal impact on either the GPU or CPU performance. To prevent overloading the MOM node with too many simultaneous *aprun* calls, we limit the number of child *aprun* calls to no more than 5. We have successfully validated a CyberShake hazard curve calculated using this co-scheduling approach (Figure 4).

Since a typical CyberShake science study requires the calculation of hundreds to thousands of hazard curves, we can use this co-scheduling approach to pipeline our computations, running our high-throughput reciprocity calculations for the previous site while performing SGT calculations for the next one, improving utilization of available resources while reducing time-to-solution.

B. AWP API Co-scheduling

Another way to use idle CPUs efficiently for co-scheduling of CyberShake simulation with post-processing is to use the AWP Application Programming Interface (API). This API allows independent modules to be interfaced to AWPg [6]. Figure 10 summarizes how AWP API works. AWPg code runs as the main thread while other modules run as parallel pthreads. The main thread initializes the individual modules, and performs wave propagation calculation. Every time a new chunk of velocity output data is copied from GPU to CPU (at specified time step instances), the main thread signals all the modules that are waiting for that data, and then continues with wave propagation calculation. Each signaled module works on the data performs whatever operation it needs to do, and returns back to waiting for the next specified time step for velocity data to be available on the CPU side. Notice that while individual modules work on the copied velocity data, the main thread continues to compute next time steps until the next specified time step instance.

The SGT calculation API module is in progress. This module is signaled when velocity data is copied from GPU to CPU. Then it calculates the strain tensors on one of the idle CPUs. These tensor results are aggregated for a specified number of time steps before writing out using collective MPI-IO.

The main difference in this tensor module and AWP-SGTg is that it computes the SGT variables on one of the idle CPUs, rather than on the GPU. While the tensor calculation on a GPU is faster because of massive parallelism, the tensor module allows overlapping of solver computation with the SGT calculation using an otherwise idle CPU.

The tensor module has memory advantages as well. AWP-SGTg must keep tensor constants and receiver coordinates in the GPU memory. However the tensor module keeps that data in XK7 node's memory, which is much less restrictive (XK7 node memory is 64 GB, compared to GPU's 6 GB memory).

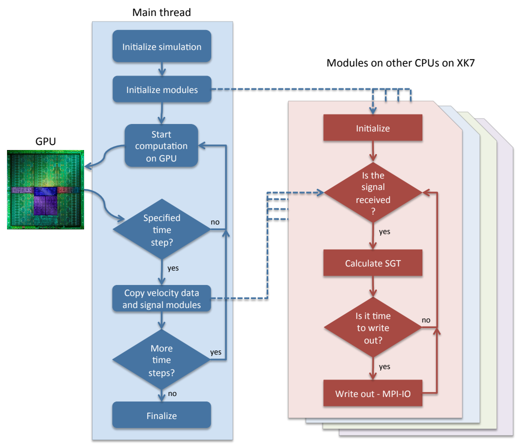


Figure 10: Flow chart of AWPg with AWP API. Main thread starts with initializations and initializes the individual modules. Then in the computational loop, GPU computes the wave propagation equations. At specified time steps, velocity data is copied from GPU to CPU and the modules are signaled. The main thread continues with the computational loop until the end of the simulation. SGT module is presented as an example. After the initializations, module waits for the signal. When the signal is received, the six SGT variables are calculated and aggregated until writing out. When the specified aggregation chunk size is reached, SGT variables are written out using collective MPI-IO.

One disadvantage of the SGT module is that it requires change in the source input reading code of AWPg to support various input modes. This code level change has minimal impact on the performance of other modules, or the main solver computation. This is because the number of related source points is much smaller than the total number of grid points, and fewer time steps are processed.

We plan to do a comprehensive performance comparison between the SGT module enabled AWPg and AWP-SGTg.

VIII. CONCLUSIONS AND OUTLOOK

We have developed a highly efficient earthquake wave propagation software to perform CyberShake strain Green tensor and post-processing calculations. This new software is designed for effective use of NCSA's hybrid BW system. Communication optimizations and scalable IO have produced a SGT solver that achieves excellent scalability on petascale systems. Our short-term goals are to increase the limit of seismic frequencies to above 1 Hz and produce a California-wide seismic hazard model using the UCERF2 rupture forecast [9]. The computational size of the statewide model will be more than 200 times larger than the current Los Angeles models. Along with CyberShake PSHA calculations, we will apply this new code for use in full-3D waveform tomography, in the development of improved ismic velocity models, which are required inputs to CyberShake. Accelerating full-3D tomography based on the current success of strain tensor calculation is our next goal. This software will provide highly scalable solutions for other problems of interest to SCEC as well as the wider scientific community to obtain better velocity models for use in structural studies of the Earth across a range of geographic scales.

ACKNOWLEDGMENT

We thank Gregory Bauer and Omar Padron of NCSA, Robert Fiedler of Cray for their major contribution of the topology related optimization and the resource support. Blue Waters computing resource was provided by NCSA through Petascale Research in Earthquake System Science on Blue Waters PRAC (Petascale Computing Resource Allocation) under NSF award number OCI-0832698. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. Computations performed on Titan at the Oak Ridge National Laboratory was supported under DOE Contract No. DE-AC05-00OR22725. Research funding was provided through UCSD Graduate Program, Blue Waters Direct Funding, SCEC's core program funding, NSF Geoinformatics award: Community Computational Platforms for Developing Three-Dimensional Models of Earth Structure (EAR-1226343), and NSF Software Environment for Integrated Seismic Modeling (OCI-1148493). This research was supported by SCEC which is funded by NSF Cooperative Agreement EAR-0529922 and USGS Cooperative Agreement 07HQAG0008. Y. Cui's work was supported through XSEDE's Extended Collaborative Support Service (ECSS) program, and L. Shih's work was supported by XSEDE Campus Champion Fellow program. SCEC contribution #1778.

REFERENCES

[1] Callaghan, S., Deelman, E., Gunter, D., Juve, G., Maechling, P., Brooks, C., Vahi, K., Milner, K., Graves, R., Field, E., Okaya, D. and Jordan, T. 2010. Scaling up workflow-based applications. *Journal of Computer and System Sciences*, 76, 6 (Sep. 2010), 428–446.

[2] Cerjan, C., Kosloff, D., Kosloff, R. and Reshef, M. 1985. A nonreflecting boundary condition for discrete acoustic and elastic wave equations. *Geophysics*, 50, 4, 705-708.

[3] Chen, P., Jordan, T. H. and Zhao, L. 2007. Full three-dimensional tomography: a comparison between the scattering- integral and adjoint-wavefield methods. *Geophysical Journal International*, 170, 1, 175-181.

[4] Chen, P., Zhao, L. and Jordan, T. H. 2007, Full 3D tomography for the crustal structure of the Los Angeles region. *Bulletin of the Seismological Society of America*, 97, 4, 1094-1120.

[5] Cui, Y., Olsen, K. B., Jordan, T. H., Lee, K., Zhou, J., Small, P., Roten, D., Ely, G., Panda, D. K., Chourasia, A., Levesque, J., Day, S. M. and Maechling, P. 2010. Scalable earthquake simulation on petascale supercomputers. *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis* (SC'10, New Orleans, November 2010), 1-20.

[6] Cui, Y., Poyraz, E., Olsen, K., Zhou, J., Withers, K., Callaghan, S., Larkin, J., Guest, C., Choi, D., Chourasia, A., Shi, Z., Day, S.,

Maechling, P. and Jordan, T., Physics-based Seismic Hazard Analysis on Petascale Heterogeneous Supercomputers, *SC13,* Denver, Nov 17-22, 2013 (submitted).

[7] Fiedler, R.A. and Whalen, S. 2012. Improving Task Placement for Applications with 2D, 3D, and 4D Virtual Cartesian Topologies on 3D Torus Networks with Service Nodes. *CUG 2013,* 1-8. Napa Valley, CA, May 6-9, 2013.

[8] Fiedler, R.A., 2013. Improving Performance of All-to-All, Random Pai, and Nearest-Neighbor Communication on Blue Waters. Presentation Slides for *PRAC Workshop*, Februrary 27, 2013.

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

[10] Field, E. H., Jordan, T. H. and Cornell, C. A. 2003. OpenSHA: A developing community-modeling environment for seismic hazard analysis. *Seismological Research Letters*, 74, 4, 406-419.

[11] Georgia Tech 2013. Keeneland User Guide. [Online]. https://www.xsede.org/gatech-keeneland.

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

[13] Nvidia GK110 Architecture Whitepaper. Online. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[14] Oak Ridge Leadership Computing Facility 2013. Titan User Guide. [Online]. https://www.olcf.ornl.gov/support/system-user-guides/titan-user-guide/.

[15] Olsen, K. B. 1994. Simulation of three-dimensional wave propagation in the Salt Lake basin. *University of Utah, Doctoral dissertation*.

[16] Schmedes, J., Archuleta, R. J. and Lavallée, D. 2010. Correlation of earthquake source parameters inferred from dynamic rupture simulations. *Journal of Geophysical Research: Solid Earth*, 115, B3.

[17] Southern California Earthquake Center. 2013. CyberShake [Online]. http://scec.usc.edu/scecpedia/CyberShake.

[18] University of Illinois NCSA 2013. Blue Waters System Overview. [Online]. https://bluewaters.ncsa.illinois.edu/user-guide.

[19] Zhao, L., Chen, P. and. Jordan, T. H. 2006. Strain Green's tensors, reciprocity, and their applications to seismic source and structure studies. *Bulletin of the Seismological Society of America*, 96, 5, 1753-1763.

[20] Zhou, J., Unat, D., Choi, D., Guest, C. an Cui, Y. 2012. Hands-on performance tuning of 3D finite difference earthquake simulation on GPU fermi chipset. In *Proceedings of International Conference on Computational Science* (ICCS'12, Omaha, Nebraska, June 4-6, 2012), 9, 976-985.

[21] Zhou, J., Cui, Y., Poyraz, E., Choi, D. and Guest, C. 2013. Multi-GPU implementation of a 3D finite difference time domain earthquake code on heterogeneous supercomputers. In *Procceding of International Conference on Computational Science* (Barcelona, June 5-7, 2013).