Parallel Finite Element Earthquake Rupture Simulations on Quad- and Hex-core Cray XT Systems

Xingfu Wu
Institute for Applied Mathematics and Computational Science
Department of Computer Science & Engineering, Texas A&M University, College Station, TX 77843
Email: wuxf@cse.tamu.edu

Benchun Duan
Department of Geology & Geophysics, Texas A&M University, College Station, TX 77843
Email: bduan@tamu.edu

Valerie Taylor
Department of Computer Science & Engineering, Texas A&M University, College Station, TX 77843
Email: taylor@cse.tamu.edu

Abstract: In this paper, we integrate a 3D mesh generator into the simulation, and use MPI to parallelize the 3D mesh generator, illustrate an element-based partitioning scheme for explicit finite element methods, and based on the partitioning scheme and what we learned from our previous work, we implement our hybrid MPI/OpenMP finite element earthquake simulation code in order to not only achieve multiple levels of parallelism of the code but also to reduce the communication overhead of MPI within a multicore node by taking advantage of the shared address space and on-chip high inter-core bandwidth and low inter-core latency. We evaluate the hybrid MPI/OpenMP finite element earthquake rupture simulations on quad- and hex-core Cray XT 4/5 systems from Oak Ridge National Laboratory using the Southern California Earthquake Center (SCEC) benchmark TPV 210. Our experimental results indicate that the parallel finite element earthquake rupture simulation obtains the accurate output results and has good scalability on these Cray XT systems.

Keywords*:* Parallel simulations, Earthquake rupture, Finite element method, OpenMP, MPI, and Multicore

1. Introduction

Numerical modeling of dynamic earthquake rupture propagation and seismic wave propagation provides seismologists with a powerful tool to discover the underlying physics controlling earthquake rupture process and resultant near-field ground motion. Figure 1 shows an example of how near-field ground motion might be controlled by rupture propagation on the fault in the 2008 Ms 8.0 Wenchuan earthquake [Duan10]. This devastating earthquake occurred in Wenchuan county, Sichuan province of China on May 12th, 2008, and killed more than 60,000 people. In Figure 1, the black line is the trace of a shallow dipping fault in the model, and circle, triangular, plus, and cross signs denote the epicenter, Chengdu, Beichuan, and Wenchuan cities, respectively. Distribution of near-field ground velocity is strongly affected by the shallow dipping fault geometry with higher ground motion on the hanging wall side of the fault (below the black line in the figure). These numerical models are also necessary to assess possible rupture scenarios in future earthquakes in earthquake-prone areas such as California, which are critical for seismic hazard analysis in these regions. Due to scarcity of near-field strong ground motion recordings, strong ground motion prediction from future earthquakes largely depends on these numerical models.

Most widely used numerical codes in the field of earthquake dynamic source models are based on the finite difference method (FDM) [AB03, SR08, CM08]. But it is difficult for FDM to deal with complex fault geometry and complex geological structures. Duan et al. [DO06, DO07, DD08] have been developing and using an explicit dynamic finite element method (EQdyna) to implement sequential simulations for modeling spontaneous earthquake rupture on geometrically complex faults, such as faults with bends, stepovers, or branches. However, a sequential simulation takes more than 120 hours for relatively small earthquake

model datasets for the Wenchuan earthquake (with ~ 46 million elements) on a SUN server with 4 dual-core AMD Opteron processors. It means waiting for five days to verify and validate a model. Therefore, it is necessary to parallelize the sequential earthquake simulation code. On one hand, the parallel earthquake simulation can significantly shorten the simulation time by fully utilizing all processor cores. On the other hand, the parallel simulation will make it feasible to utilize large-scale supercomputing resources from TAMU supercomputing facilities and other national labs.

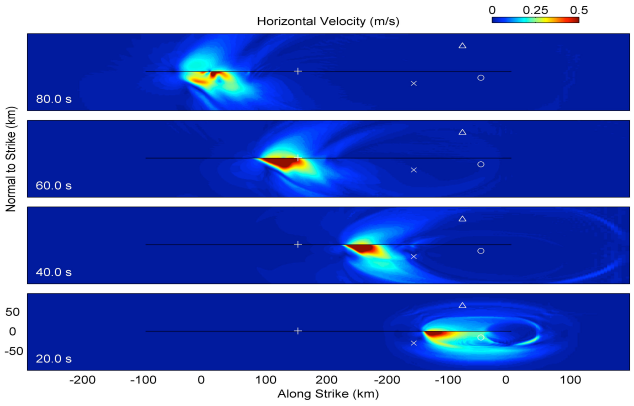

Figure 1. Snapshots of horizontal ground velocity from a simplified dynamic model of the 2008 Ms 8.0 Wenchuan earthquake [Duan10]

In the finite element method, the data dependence is much more irregular than the finite difference method, so it is generally more difficult to parallelize. Ding and Ferraro [DF96] discussed node-based and element-based partitioning strategies, found that main advantage for element-based partitioning strategy over node-based partitioning strategy was its modular programming approach to the development of parallel applications, and developed an element-based concurrent partitioner for partitioning unstructured finite element meshes on distributed memory architectures.

Mahinthakumar and Saied [MS02] presented a hybrid implementation adapted for an implicit finite-element code developed for groundwater transport simulations based on the original MPI code using a domain decomposition strategy, and added OpenMP directives to the code to use multiple threads within each MPI process on SMP clusters. Nakajima [NK03] presented a parallel iterative method in GeoFEM for finite element method which was node-based with overlapping elements on the Earth Simulator, and explored a three-level hybrid parallel programming model, including message passing (MPI) for inter-SMP node communication, loop directives by OpenMP for intra-SMP node parallelization and vectorization for each processing element.

In our previous work [WD09, WD11], we used OpenMP to parallelize a sequential earthquake simulation code EQdyna for modeling spontaneous dynamic earthquake rupture along geometrically complex faults, and based on what we learned from the OpenMP implementation, we developed an initial hybrid MPI/OpenMP implementation of the sequential earthquake simulation code EQdyna with a 3D mesh as an input, which was generated by a 3D mesh generator separately before the simulation execution. In this paper, we integrate the 3D mesh generator into the simulation, and use MPI to parallelize the 3D mesh generator, illustrate an element-based partitioning scheme for explicit finite element methods, and evaluate its performance on Quad- and Hex-core Cray XT systems at Oak Ridge National Laboratory [NCCS] using the Southern California Earthquake Cente (SCEC) benchmark TPV 210. The experimental results indicate that the hybrid MPI/OpenMP implementation has the accurate output results and the good scalability on these systems.

The remainder of this paper is organized as follows. Section 2 illustrates an element-based partitioning scheme, discusses our hybrid MPI/OpenMP parallel finite element Earthquake rupture simulations in detail. Section 3 describes the architecture and memory hierarchy of quad- and hex-core Cray XT systems used in our experiments. Section 4 discusses the benchmark problem TPV210 and verifies our simulation results. Section 5 evaluates and explores performance characteristics of our hybrid MPI/OpenMP implementation, and presents the experimental results. Section 6 concludes this paper.

2. Hybrid MPI/OpenMP Parallel Finite Element Earthquake Rupture Simulations

In this section, based on what we learned from our previous work [WD09, WD11], we integrate a 3D mesh generator into the simulation, and use MPI to parallelize the 3D mesh generator, illustrate an element-based partitioning scheme for explicit finite element methods, and discuss how efficiently to use hybrid MPI/OpenMP implementations in the earthquake simulations for not only achieving multiple levels of parallelism but also reducing the communication overhead of MPI within a multicore node, by taking advantage of the globally shared address space and on-chip high inter-core bandwidth and low inter-core latency on large-scale multicore systems.

2.1 Mesh Generation and Model Domain Partitioning

In our previous work [WD09, WD11], we developed an initial hybrid MPI/OpenMP implementation of the sequential earthquake simulation code EQdyna with a 3D mesh as an input, which was generated by a 3D mesh generator separately before the simulation execution. As we discussed in our previous work, the earthquake simulation code is memory bound, when the number of elements

increases, the required system memory for storing large arrays associated with the entire model domain increases dramatically. In order to overcome the limitation, in this paper, we integrate the 3D mesh generator into the simulation, and use MPI to parallelize the 3D mesh generator.

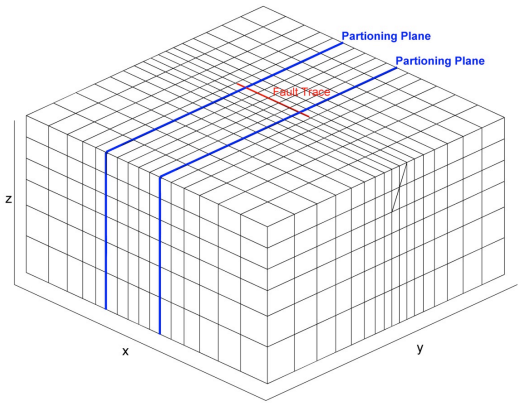

Figure 2. Schematic diagram to show mesh and model domain partitioning

To parallelize the 3D mesh generator, based on the number of MPI processes used, we partition the entire model domain by the coordinate along fault strike (e.g., the x-coordinate in a Cartesian coordinate system) shown in Figure 2 so that we can define small arrays for each MPI process independently. Figure 2 gives a schematic diagram for the 3D mesh partitioning. Thus, memory requirements by large arrays that are associated with the entire model domain in a previous version of the code [WD11] significantly decrease.

To facilitate message passing between adjacent MPI processes, based on the partitions of the entire model domain by the coordinate along fault strike, during the mesh generation step, we create a sub-mesh for each MPI process and record shared boundary nodes between two adjacent MPI processes. This converts reading initial large input mesh data to computing and generating small mesh data for each MPI process. Note that, in this partitioning scheme, the maximum number of MPI processes that can be used is bounded by the total number of nodes along the x-coordinate.

2.2 Element-based Partitioning

In our explicit finite element earthquake simulation, we primarily use trilinear hexahedral elements to discretize a 3D model for computational efficiency, with wedge-shaped elements along the fault to characterize dipping fault geometry as illustrated in Figure 2. We use a large buffer region with increasingly coarser element sizes away from the fault to prevent reflections from artificial model boundaries from contaminating examined phenomena.

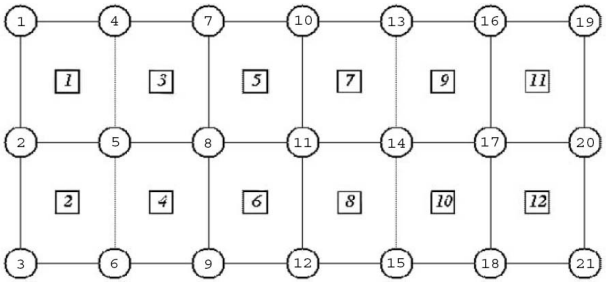

Figure 3. 2D geometry for the EQdyna: 12 elements (boxes) and each element with 4 nodes (circles)

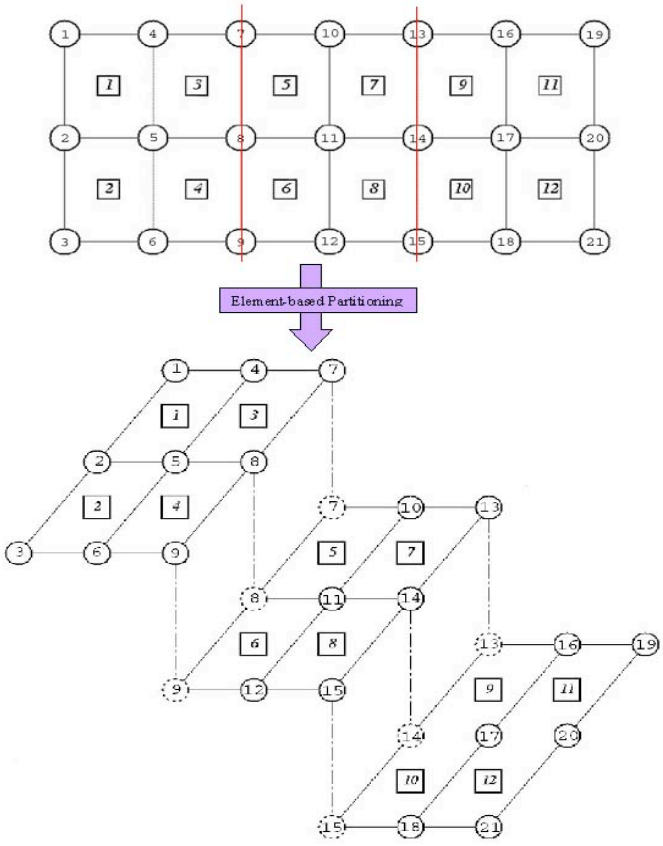

Figure 4. Element-based Partitioning Scheme

For simplicity, we discuss our partitioning scheme with a hypothetical 2D mesh shown in Figure 3, where there are 12 elements (boxes) and each element has four nodes (circles) adjacent to it. We propose an element-based partitioning scheme because most time-consuming computation in the earthquake rupture simulation code is element-based. Within one timestep, element contribution (both internal force and hourglass force) to its nodes' nodal force is first calculated. Then, contributions to a node's nodal force from

all of its adjacent elements are assembled. For instance, the nodal force at node 1 only involves element 1, while the nodal force at node 5 involves elements 1, 2, 3, and 4. The nodal force at node 5 is the sum of contributions from all these four elements.

Figure 4 illustrates the element-based partitioning scheme for the finite element method, where the 2D domain is split into three components. In this scheme, we essentially partition the model domain based on element numbers. Each component consists of four elements and the nodes adjacent to them. A node that lies on the boundary between two components is called a boundary node. For example, nodes 7, 8 and 9 are the boundary nodes between the first two components, and nodes 13, 14 and 15 are the boundary nodes between the last two components. To update the nodal force at a boundary node such as node 8, it needs contributions from elements 3 and 4 in the first component and those from elements 5 and 6 in the second component. This requires the data exchange between the first two components.

Similarly, the above element-based partitioning scheme can be extended to large 3D datasets. The element-based partitioning method described in this section is applicable to more irregular meshes as well.

2.3 Hybrid Implementations

Multicore clusters provide a natural programming paradigm for hybrid programs. Generally, MPI is considered optimal for process-level coarse parallelism and OpenMP is optimal for loop-level fine grain parallelism. Combining MPI and OpenMP parallelization to construct a hybrid program is not only to achieve multiple levels of parallelism but also to reduce the communication overhead of MPI at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth contention. Therefore, we use hybrid MPI/OpenMP to parallelize the finite element code for exploring the parallelism of the code at node level (OpenMP) and the parallelism of the code between nodes (MPI) so that the parallel earthquake simulation can be run on most supercomputers. Note that, in the hybrid MPI/OpenMP implementations, we separate MPI regions from OpenMP regions, and OpenMP threads cannot call MPI subroutines.

Figure 5 shows the parallelism at MPI and OpenMP levels within one timestep for the hybrid implementation of the earthquake simulation. As we discussed in the previous section, using the element-based partitioning scheme, we can partition the 2D mesh geometry into three components, and dispatch each component to a MPI process for MPI level parallelism. So each MPI process is in charge of four elements and the nodes adjacent to them. Because the earthquake simulation is memory-bound, each MPI process is created on a different node as illustrated in Figure 5. MPI process 0 is run on Node 0; process 1 is on Node 1; process 2 is on Node2. On each node, OpenMP level parallelism can be achieved by using element-based partitioning scheme and OpenMP. Each MPI process (the master thread) forks several new threads to take advantage of the shared address space and on-chip high inter-core bandwidth and low inter-core latency on the node.

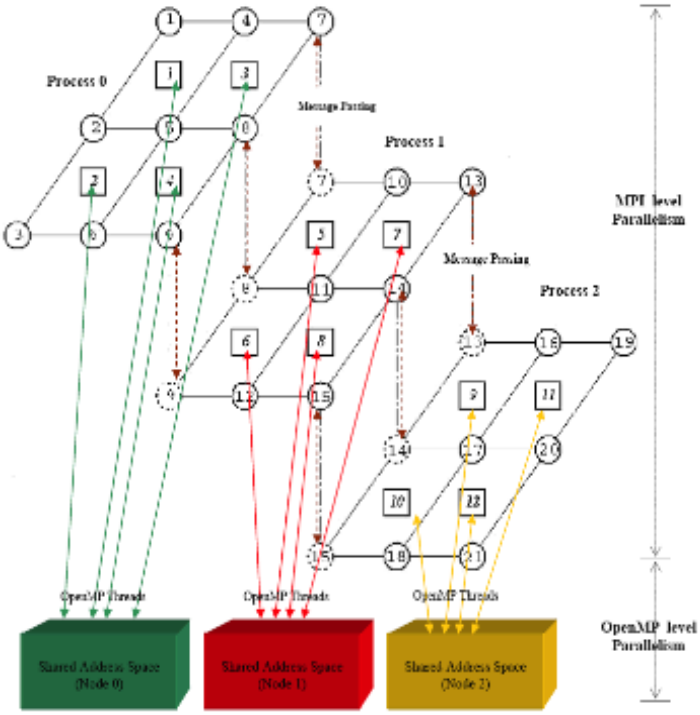

Figure 5. Parallelism at MPI and OpenMP levels within one timestep

To manipulate and update nodal forces at these boundary nodes, it requires the data exchange between two MPI processes via message passing. For each boundary node such as node 7 shown in Figure 5, to update its nodal force at the end of each timestep, we sum the nodal force at node 7 from process 0 and the nodal force at node 7 from process 1, then use the sum to update the nodal forces at node 7 for the processes 0 and 1.

To implement updating the nodal force at each boundary node at the end of each timestep, we propose the following algorithm to deal with the problem.

***Algorithm**: Update the nodal forces at boundary nodes:*

***Step 1**: Partition the initial data mesh based on the number of MPI processes to ensure load balancing, get the information about shared boundary nodes between MPI processes i and i+1 from the mesh generator discussed in Section 2.1, and allocate a temporal array btmp with the nodal forces at the shared boundary nodes,*

***Step 2**: The MPI process i sends the array btmp to its neighbor process i+1 using MPI_Sendrecv,*

***Step 3**: The MPI process i+1 receives the array from process i using MPI_Sendrecv. For each shared boundary node, it sums the nodal force from the array and the local nodal force at the shared boundary node, then assigns the summation to the nodal force at the shared node locally,*

***Step 4**: The MPI process i+1 updates the array locally, and sends the updated array back to the MPI process i,*

***Step 5**: The MPI process i receives the updated array and update the nodal forces at the shared nodes locally, and deallocates the temporal array at the end of the timestep*

***Step 6**: Repeat the above Steps 1-5 for the next timestep.*

The algorithm implements the straightforward data exchanges illustrated in Figure 5, and it is efficient because of sending/receiving smaller messages. This can simplify the programming efforts and reduce the communication overhead.

3 Experimental Platforms

In this paper, we conduct our experiments using Jaguar (Cray XT5 and XT4) from Oak Ridge National Laboratory [NCCS]. Table 1 shows their specifications and the same compilers used for all experiments. All systems have private L1 and L2 caches and shared L3 cache per node. Jaguar is the primary system in the ORNL Leadership Computing Facility (OLCF). It consists of two partitions: XT5 and XT4 partitions shown in Figure 6.

Table 1. Specifications of quad- and hex-core Cray XT Systems

Configurations	JaguarPF (XT5)	Jaguar (XT4)
Total Cores	224,256	31,328
Total Nodes	18,688	7,832
Cores/Socket	6	4
Cores / Node	12	4
CPU type	AMD 2.6GHz hex-core	AMD 2.1GHz quad-core
Memory/Node	16GB	8GB
L1 Cache/Core, private	64 KB	64 KB
L2 Cache/Core, private	512KB	512KB
L3 Cache/Socket, shared	6MB	2MB
Compiler	ftn	ftn
Compiler Options	-O3 -mp=nonuma -fastsse	-O3 -mp=nonuma -fastsse

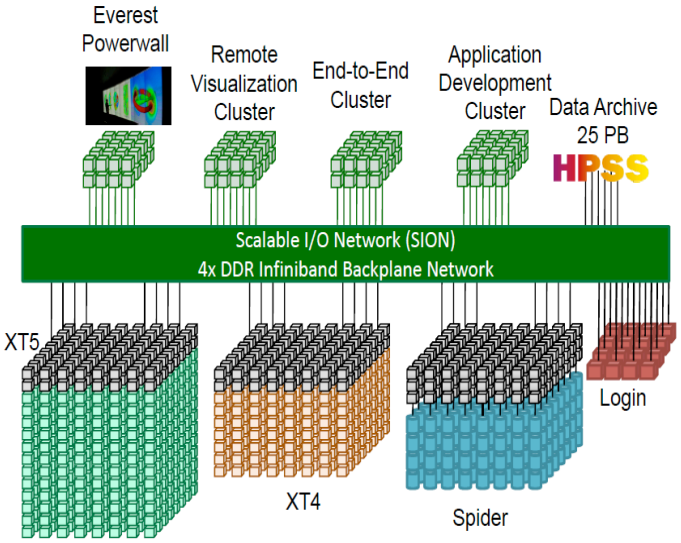

Figure 6. Jaguar and JaguarPF System Architecture [NCCS]

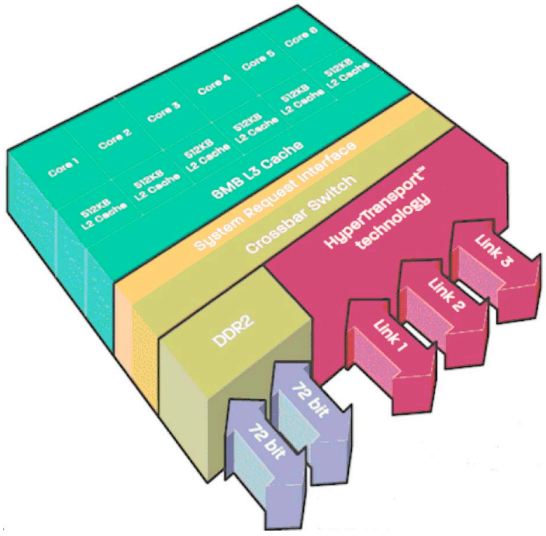

Figure 7. AMD hex-core Opteron chip architecture [NCCS]

The Jaguar XT5 partition (JaguarPF) contains 18,688 compute nodes in addition to dedicated login/service nodes. Each compute node contains dual hex-core AMD Opteron 2435 (Istanbul shown in Figure 7) processors running at 2.6GHz, 16GB of DDR2-800 memory, and a SeaStar 2+ router. The resulting partition contains 224,256 processing cores, 300TB of memory, and a peak performance of 2.3 petaflop/s. The Jaguar XT4 partition (Jaguar) contains 7,832 compute nodes in addition to dedicated login/service nodes. Each compute node contains a quad-core AMD Opteron 1354 (Budapest) processor running at 2.1 GHz, 8 GB of DDR2-800 memory, and a SeaStar2 router. The resulting partition contains 31,328 processing cores, more than 62 TB of memory, over 600 TB of disk space, and a peak

performance of 263 teraflop/s. The SeaStar2+ router (XT5 partition) has a peak bandwidth of 57.6GB/s, while the SeaStar2 router (XT4 partition) has a peak bandwidth of 45.6GB/s. The routers are connected in a 3D torus topology, which provides an interconnect with very high bandwidth, low latency, and extreme scalability.

4. Result Verification and Benchmark Problems

4.1 Benchmark Problem TPV210

To validate the hybrid MPI/OpenMP earthquake simulation code, we apply it to a SCEC/USGS benchmark problem TPV210, which is the convergence test of the benchmark problem TPV10 [HB09, SUVP]. In TPV10, a normal fault dipping at 60° (30 km long along strike and 15 km wide along dip) is embedded in a homogeneous half space. Pre-stresses are depth dependent and frictional properties are set to result in a subshear rupture. This benchmark problem is motivated by ground motion prediction at Yucca Mountain, Nevada, which is a potential high-level radioactive waste storage site [DD10]. In TPV10, modelers are asked to run simulations at an element size of 100 m on the fault surface. In TPV210, we conduct the convergence test of the solution by simulating the same problem at a set of element sizes, i.e., 200 m, 100 m, 50 m, 25 m, 12.5 m, and so on. Here, we work on 100 m and 50 m element sizes. Table 2 summarizes model parameters for the two element sizes. Because the number of elements with a discretization varies a little bit with the number of MPI processes used, Table 2 only gives a rough estimate of this number. In the table, nxt is the node number along the x-coordinate in a sequential simulation, which limits how many MPI processes one can use in a hybrid parallel simulation.

Table 2. Model parameters for two element sizes

Parameters	TPV210-100m	TPV210-50m
Element size	100 m	50 m
Total elements	~ 25,000,000	~100,000,000
Time step (sec)	0.008	0.004
Termination Time (seconds)	15	15
nxt	477	829

4.2 Result Verifications

Figure 8 show the rupture time (in seconds) contours on the 60° dipping fault plane. Red star denotes the hypocenter of simulated earthquakes. Results from two simulations are plotted in the figure. One (black) is the result from a previous run with 50 m element size that was verified in the SCEC/USGS code validation exercise [SUVP]. The other is the result from a run performed in this study on JaguarXT5 with 50 m element size using 256 MPI processes. These two results essentially overlap, indicating our current hybrid implementation gives accurate results.

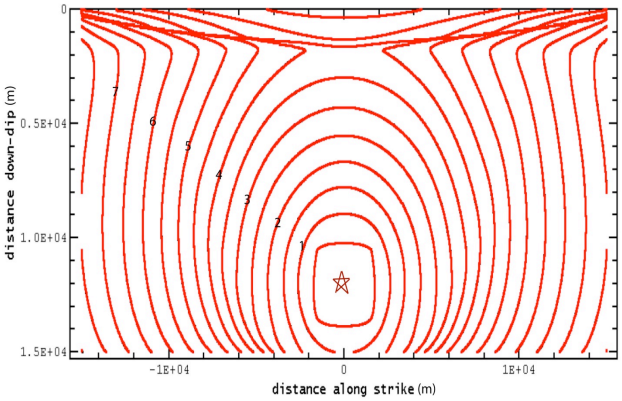

Figure 8. Rupture time contours on the dipping fault plane for TPV210 with 50m

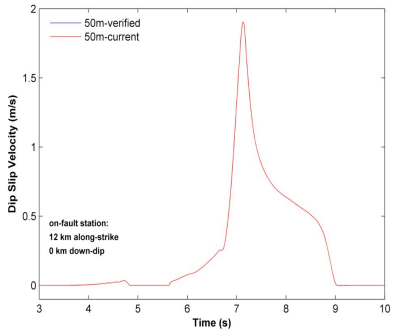

Figure 9. The dip-slip component of slip velocity on a fault station for TPV210 with 50m

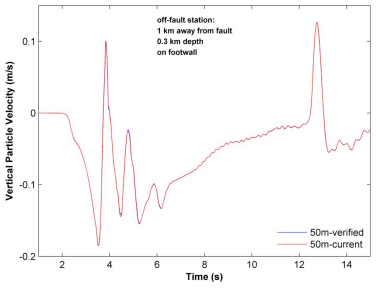

Figure 10. The vertical component of particle velocity at an off-fault station for TPV210 with 50m

Figures 9 and 10 compares time histories of the dip-slip component of slip velocity at an on-fault station and the vertical component of particle velocity at an off-fault station from the two simulations with 50m discussed above. The locations of the stations are in the figures. The result from the current hybrid implementation matches that from the verified one very well. This indicates that our hybrid MPI/OpenMP implementation is validated and has the accurate output results of fault movement and ground shaking.

5. Performance Analysis and Comparison

In this section, we analyze and compare the performance of the hybrid MPI/OpenMP finite element earthquake simulation on quad- and hex-core Cray XT systems. Note that **TPN** stands for **Threads Per Node**.

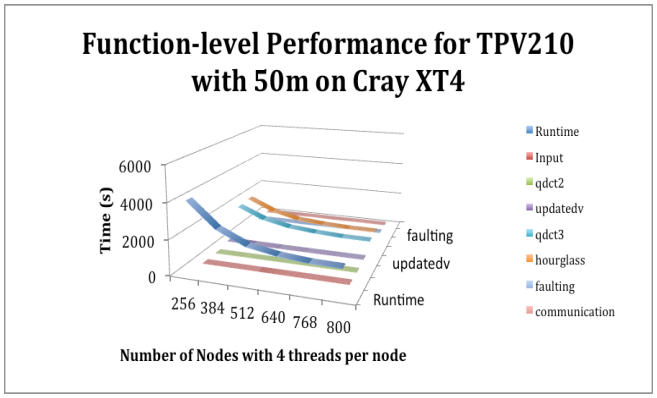

Figure 8. Function-level performance for TPV210 with 50m on Cray XT4

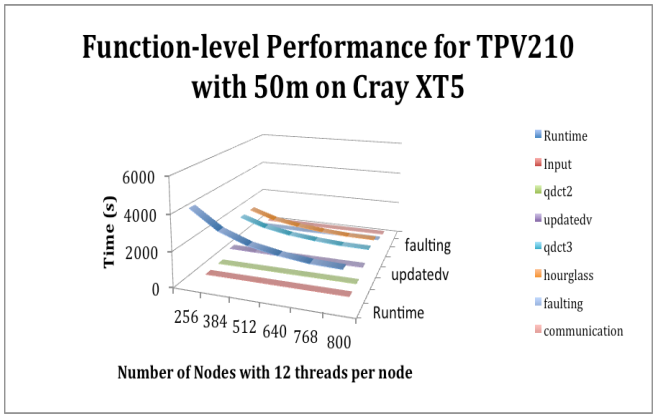

Figure 9. Function-level performance for TPV210 with 50m on Cray XT5

Figures 8 and 9 presents the function-level performance of the hybrid MPI/OpenMP finite element earthquake simulation with 50m on Cray XT4 and XT5 systems, where there are seven main functions in the code; the functions Input and qdct2 are called once, and the functions updatedv, qdct3, hourglass, faulting and communication are within the main timestep loop. The function communication means the MPI communication, and the MPI communication overhead was measured on each master MPI process for all hybrid executions.

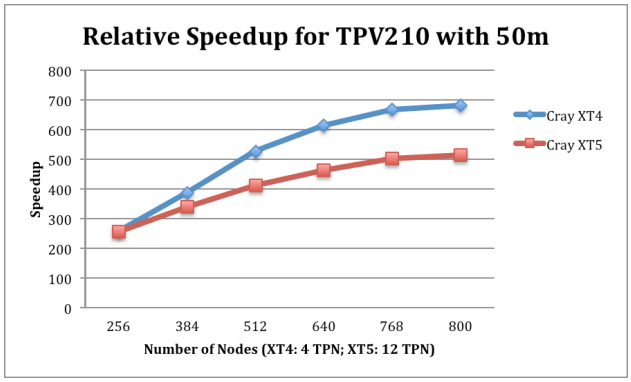

Figure 10. Relative Speedup for TPV 210 with 50m

Figure 10 shows the relative speedup for TPV210 with 50 m on Cray XT4 and XT5 systems from Figures 8 and 9, where we assume that the relative speedup for TPV210 with 50 m executed on 256 nodes is 256, then calculate the relative speedup for 384 to 800 nodes. In fact, for 256 nodes, the hybrid execution on Cray XT4 utilizes 256 MPI processes with 1 MPI process per node and 4 OpenMP TPN; the hybrid execution on Cray XT5 utilizes 256 MPI processes with 1 MPI process per node and 12 OpenMP TPN. We observe that the hybrid execution on Cray XT4 has better scalability than that on Cray XT5. From Table 1, Cray XT5 is much faster than Cray XT4. As shown in Figures 8 and 9, the MPI communication overhead was similar on Cray XT4 and XT5 systems. What really caused the performance degradation on the Cray XT5 system?

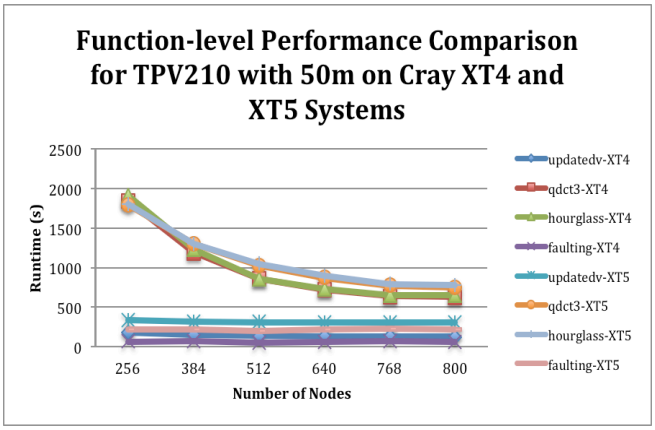

Figure 11. Function-level Performance Comparison for TPV210 with 50m

Table 3. Percentage difference for updatedv (of TPV210 with 50m) on Cray XT4 and XT5

Number of Nodes	updatedv-XT4 (s)	updatedv-XT5 (s)	% difference
256	176.87	339.29	91.83
384	145.91	321.53	120.36
512	133.83	313.45	134.22
640	128.82	308.36	139.37
768	125.97	305.27	142.34
800	125.89	305.13	142.38

Figure 11 indicates that four main functions updatadv, qdct3, hourglass and faulting are the primary source of performance degradation, where updatadv-XT4 means the performance of updatedv on Cray XT4, and updatedv-XT5 means the performance of updatedv on Cray XT5, and so on. The hybrid executions on Cray XT4 and Cray XT5 are the same except the difference in number of OpenMP threads per node. We used 4 OpenMP threads per node for Cray XT4, and 12 OpenMP threads per node for Cray XT5. How did this impact the performance? From Figure 11, the function updatedv has the biggest percentage difference up to 142.38% shown in Table 3. The function updatedv entails updating velocity and displacement at each time step. Its code segment is as follows:

```
!$omp parallel do default(shared) private(l,j,k)
  do l=1,numnp
    do j=1,ndof
      k=id(j,l)
      if(k > 0) then        !only non-boundary,update
        v(j,l) = v(j,l) + brhs(k) * dt
        d(j,l) = d(j,l) + v(j,l) * dt
      endif
    enddo
  enddo
  !$omp end parallel do
```

Where numnp is the total number of nodes assigned to each MPI process. Because we use element-based partitioning, the total number of nodes are assocated with the number of elements assigned to each MPI process. From Table 3, we can observe that using 12 OpenMP threads per MPI process on Cray XT5 has more OpenMP overhead than using 4 OpenMP threads per MPI process on Cray XT4. The similar performance trend occurs for other functions. This is similar to what we found in [WT11], the number of OpenMP threads per node for hybrid programs is limited by number of cores per node in the underlying system, the underlying system software, as well as the loop size to which OpenMP parallelization is applied. For strong scaling scientific applications like our hybrid earthquake simulation, with increasing number of cores, some parallelization loop sizes become very small, which may cause more OpenMP overhead.

The other reason is related to memory subsystems and how efficiently they support OpenMP programming. In the function updatedv, we use the default OpenMP scheduling, i.e, static scheduling without chunck, which distributes iterations in blocks of size approximately *numnp/12* for Cray XT5 and *numnp/4* for Cray XT4 over all threads in a round-robin fashion, thus, using 12 OpenMP threads per MPI process on Cray XT5 caused more OpenMP load imbalance than using 4 OpenMP threads per MPI process on Cray XT4.

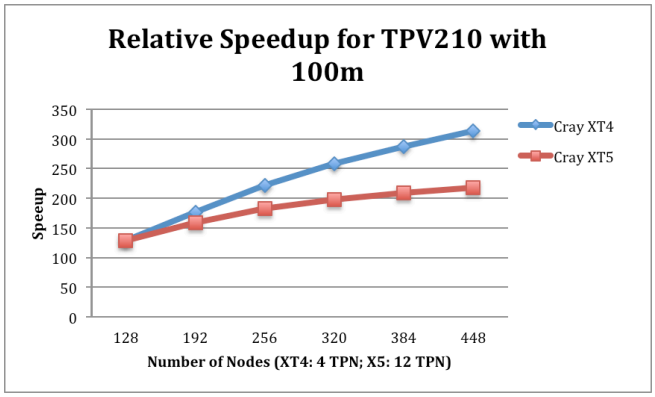

Figure 12. Relative Speedup for TPV 210 with 100m

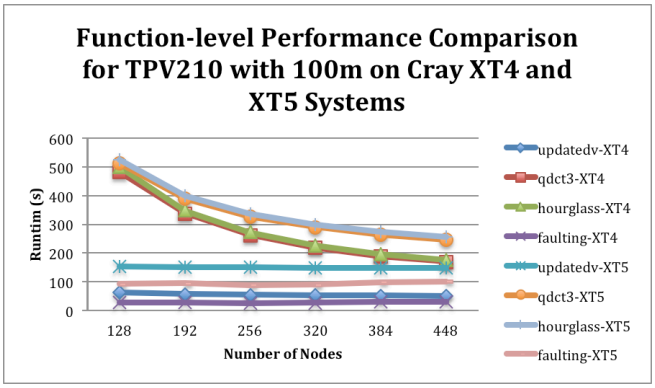

Figure 13. Function-level Performance Comparison for TPV210 with 100m

Similarly, we find the same performance trend for TPV210 with 100m on Cray XT4 and XT5 systems shown in Figures 12 and 13.

6. Conclusions

In this paper, we integrated a 3D mesh generator into the simulation, and used MPI to parallelize the 3D mesh generator, illustrated an element-based partitioning scheme for explicit finite element methods, and based on the

partitioning scheme and what we learned from our previous work, we implemented our hybrid MPI/OpenMP finite element earthquake simulation code to achieve multiple levels of parallelism of the code. The experimental results demonstrated that the hybrid MPI/OpenMP implementation has the accurate output results and the good scalability on Cray XT4 and XT5 systems. However, for the benchmark problem TPV 210, it is interesting to observe that using 12 OpenMP threads per MPI process on Cray XT5 has more OpenMP overhead than using 4 OpenMP threads per MPI process on Cray XT4 for the hybrid executions on the same number of nodes with 1 MPI process per node although Cray XT5 is much faster than Cray XT4.

Because we partitioned the entire model domain by the coordinate along fault strike (e.g., the x-coordinate in a Cartesian coordinate system), the maximum number of MPI processes that can be used is bounded by the total number of nodes along the x-coordinate. This limits the scalability of the hybrid simulation. We also found that we could not use any number of MPI processes for the hybrid execution because load imbalance could cause large MPI communication overhead. For the future work, we plan to further improve the memory requirements of the hybrid simulation code by partitioning the entire model domain in X-, Y- and Z-dimensions, and consider some load balancing strategies discussed in [TS01].

Acknowledgements

This work is in part supported by NSF grant CNS-0911023, the Award No. KUS-I1-010-01 made by King Abdullah University of Science and Technology (KAUST), and NSF grant EAR-1015597. The authors would like to acknowledge National Center for Computational Science at Oak Ridge National Laboratory for the use of Jaguar and JaguarPF under DOE INCITE project "Performance Evaluation and Analysis Consortium End Station".

About the Authors

Xingfu Wu has been working at Texas A&M University as TEES Research Scientist since July 2003. He is a senior ACM member and an IEEE-CS member. His research interests are performance evaluation and modeling, parallel programming environments and tools, parallel scientific computing, and power and energy analysis in HPC systems. His monograph: Performance Evaluation, Prediction and Visualization of Parallel Systems, was published by Kluwer Academic Publishers (ISBN 0-7923-8462-8) in 1999.

Benchun Duan is an assistant professor in the Department of Geology & Geophysics at Texas A&M University. His research interests include earthquake rupture dynamics, seismic wave propagation, geomechanical modeling, and parallel scientific computing. He has been an active participant in the dynamic code validation exercise sponsored by the Southern California Earthquake Center and the U.S. Geological Survey.

Valerie E. Taylor earned her B.S. in Electrical and Computer Engineering and M.S. in Computer Engineering from Purdue University in 1985 and 1986, respectively, and a Ph.D. in Electrical Engineering and Computer Science from the University of California, Berkeley, in 1991. From 1991 through 2002, Dr. Taylor was a member of the faculty in the Electrical and Computer Engineering Department at Northwestern University. Dr. Taylor joined the faculty of Texas A&M University as Head of the Dwight Look College of Engineering's Department of Computer Science in January of 2003, and is, also, currently a holder of the Royce E. Wisenbaker Professorship. Her research interests are in the area high performance computing. She has authored or co-authored over 100 papers in these areas. Dr. Taylor is a member of ACM and Senior Member of IEEE-CS.

References

[AB03] V. Akcelik, J. Bielak, G. Biros, et al., High Resolution Forward and Inverse Earthquake Modeling on Terascale Computers, *SC03*, 2003.

[CM08] Y. Cui, R. Moore, K. Olsen, et al., Toward Petascale Earthquake Simulations, *Acta Geotechnica*, DOI 10.1007/s11440-008-0055-2, 2008.

[DD08] B. Duan and S. M. Day, Inelastic Strain Distribution and Seismic Radiation From Rupture of a Fault Kink, *J. Geophys. Res.*, 113, 2008.

[DD10] B. Duan and S. M. Day, Sensitivity study of physical limits of ground motion at Yucca Mountain, *Bull. Seism. Soc. Am.,* 100 (6), 2996-3019, 2010.

[DF96] H. Ding and R. Ferraro, An Element-based Concurrent Partitioned for Unstructured Finite Element Meshes, *IPPS'96,* 1996.

[DO06] B. Duan and D. D. Oglesby, Heterogeneous Fault Stresses From Previous Earthquakes and the Effect on Dynamics of Parallel Strike-slip Faults, *J. Geophys. Res.*, 111, 2006.

[DO07] B. Duan and D. D. Oglesby, Nonuniform Prestress From Prior Earthquakes and the effect on Dynamics of Branched Fault Systems, *J. Geophys. Res.*, 112, 2007.

[Duan10] B. Duan, Role of initial stress rotations in rupture dynamics and ground motion: A case study with Implications for the Wenchuan earthquake, *J. Geophys. Res.*, 115, 2010.

[HB09] R. A. Harris, M. Barall, et al., The SCEC/USGS Dynamic Earthquake-rupture Code Verification Exercise, *Seismol. Res. Letts.*, Vol. 80, No. 1, 2009.

[NCCS] NCCS Jaguar and JaguarPF, Oak Ridge National Laboratory, http://www.nccs.gov/computing-resources /jaguar/

[MS02] G. Mahinthakumar and F. Saied, A Hybrid MPI-OpenMP Implementation of An Implicit Finite-Element Code on Parallel Architectures, *the International*

Journal of High Performance Computing Applications, Vol. 16, No. 4, 2002.

[NK03] K. Nakajima, OpenMP/MPI Hybrid vs. Flat MPI On the Earth Simulator: Parallel Iterative Solvers for Finite Element Method, *ISHPC2003, LNCS 2858*, 2003.

[SR08] S. Schlosser, M. Ryan, R. Taborda, J. Lopez, D. O'Hallaron and J. Bielak, Materialized Community Groud Models for Large-scale Earthquake Simulation, *SC08*, 2008.

[SUVP] The SCEC/USGS Spontanous Rupture Code Verification Project, http://scecdata.usc.edu/cvws.

[TS01] Valerie Taylor, E. Schwabe, B. Holmer, and M. Hribar, Balancing Load versus Decreasing Communication: Parameterizing the Tradeoff, *Journal of Parallel and Distributed Computing*, Vol. 61, 567-580, 2001.

[WD09] Xingfu Wu, Benchun Duan and Valerie Taylor, An OpenMP Approach to Modeling Dynamic Earthquake Rupture Along Geometrically Complex Faults on CMP Systems, *ICPP2009 SMECS Workshop*, September 22-25, 2009, Vienna, Austria.

[WD11] Xingfu Wu, Benchun Duan and Valerie Taylor, Parallel simulations of dynamic earthquake rupture along geometrically complex faults on CMP systems, *J. Algorithm and Computational Technology,* 5 (2), 313-340, 2011.

[WT09] Xingfu Wu, Valerie Taylor, Charles Lively and Sameh Sharkawi, Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters, *Scalable Computing: Practice and Experience*, Vol. 10, No. 1, 2009.

[WT11] Xingfu Wu and Valerie Taylor, Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on Large-Scale Multicore Supercomputers, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38, Issue 4, March 2011.