Speeding Up Finite Element Wave Propagation for Large-Scale Earthquake Simulations

Ricardo Taborda
`rtaborda@cmu.edu`

Julio López
`jclopez@cs.cmu.edu`

Haydar Karaoglu
`hkaraogl@andrew.cmu.edu`

John Urbanic
`urbanic@psc.edu`

Jacobo Bielak
`jbielak@cmu.edu`

Computational Seismology Laboratory, Civil and Environmental Engineering.
Parallel Data Laboratory, Computer Science Department.

CMU-PDL-10-109
August 2010

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

This research was sponsored in part by The Gordon and Betty Moore Foundation, The Petascale Data Storage Institute (PDSI), the support of the companies of the Parallel Data Laboratory Consortium (PDL) and through NSF awards: Towards Petascale Simulation of Urban Earthquake Impacts (NSF OCI-0749227); SCEC Petascale Cyberfacility for Physics-Based Seismic Hazard Analysis (PetaSHA-2) (EAR-0122464); SCEC Enabling Earthquake System Science Through Petascale Calculations (PetaShake) (NSF OCI-749313); and SCEC ACCESS Program: Advancement of Cyberinfrastructure Careers through Earthquake System Science (SCEC/ACCESS-G). SCEC is funded through NSF Cooperative Agreement EAR-0106924 and USGS Cooperative Agreement 02HQAG0008. This research was supported by an allocation of advanced computing resources supported by the National Science Foundation and the TeraGrid Advanced Support Program. The computations were performed on Kraken at the National Institute for Computational Sciences (NICS). The SCEC contribution number for this paper is 1416. We thank R. Glenn Brook and Kwai Wong at NICS for their excellent assistance in making possible the capability runs in Kraken.

Keywords: Earthquake Ground-Motion Modeling, Finite-Element Method (FEM), Numerical Simulations, Efficient FEM

Abstract

This paper describes the implementation and performance of a new approach to finite element earthquake simulations that represents a speedup factor of 3x in the total solving time employed by *Hercules*—the octree-based earthquake simulator developed by the Quake Group at Carnegie Mellon University. This gain derives from applying an efficient method for computing the stiffness contribution at the core of the solving algorithm for the discretized equations of motion. This efficient method is about 5 times faster than our previous conventional implementation. We evaluate the performance and scalability of the new implementation through numerical experiments with the 2008 Chino Hills earthquake under various problem sizes and resource conditions on up to 98K CPU cores, obtaining excellent results. These experiments required simulations with up to 11.6 billion mesh elements. The newly obtained efficiency reveals that other areas in Hercules, such as inter-processor communication, waiting time, and additional computing processes become more critical, and that improvements in these areas will result in significant enhancement in overall performance. This latest advance has enormous implications for saving CPU hours and catapults the potential of Hercules to target larger and more realistic problems, taking full advantage of the new generation of petascale supercomputers.

1 Introduction

High-performance supercomputers are paramount in science and engineering for addressing many of the most challenging problems being studied today. They stand as unparalleled tools. Yet, with the impending advent of a new generation of supercomputers open for public research expected to surpass the hundreds of thousand processors, questions are being raised about whether existing codes will scale efficiently to fully take advantage of these resources.

This becomes even more important if one considers the amount of resources necessary for the deployment and maintenance of supercomputers. Given the fact that the smallest reduction in execution time may save thousands of allocated CPU hours, researchers are being urged—and rightly so—to make the best possible use of these resources. Making more efficient use of capability supercomputers will ultimately allow scientists to solve the most challenging problems at a lower computational and economic cost.

This paper addresses these issues in the context of the problem of earthquake simulations at a regional scale. The term *regional scale* in computational seismology refers to simulation domains of the order of hundreds of kilometers in each direction. In this kind of simulations, the earthquake source and all subregions and urban areas of interest are included within the model. Our simulations are carried out using *Hercules*, the octree-based finite element earthquake simulator developed by the Quake Group at Carnegie Mellon University (CMU) [45, 46].

We present the implementation in Hercules of an efficient method to obtain the stiffness contribution to the solution of the discretized equations of motion. This new approach is 5 times faster than the conventional method used in finite elements. Such a reduction is crucial because the computation of the stiffness contribution may account for up to 90% of the total solving time. The new method reduces this share to about 50% and results in an overall speedup factor of 3x in Hercules' total solving time.

This will allow us to tackle increasingly larger and more realistic problems at finer mesh resolutions. In earthquake simulations this means computing the response of the ground motion at higher frequencies and lower shear wave velocities. With this new advancement, Hercules sees its potential dramatically increased. Ongoing efforts will soon let us include other areas of research such as nonlinear soil behavior and urban seismology, which will truly demand the level of compute cycles of tomorrow's capability machines, fully exploiting their processing power.

We start by briefly reviewing the state of the art in earthquake simulations, where Hercules, even prior to the new developments presented here, has held a leading role. A section follows with a high-level description of the newly implemented efficient method and its relevance with respect to the overall simulation process. A detailed description of the efficient method with respect to the conventional approach is included in the Appendix. We continue with an evaluation of Hercules' performance and scalability on up to 98K processor cores, using the 2008 Chino Hills earthquake as a testbed, with emphasis on the relative differences between the conventional and the efficient methods. We conclude with a discussion of potential areas for further improving our simulation capabilities, and research related areas toward petascale simulation of regional and urban earthquake impacts.

2 Earthquake Simulations at Scale

Earthquake simulations are needed to understand the propagation of seismic waves and the ground motion during strong shaking in earthquakes-prone regions. They constitute a necessary complement to seismic recorded data. Deterministic earthquake simulations entail obtaining the solution of the linear momentum equation, shown in (1) for Cartesian coordinates. σ_{ij} represents the Cauchy stress tensor, ρ is the mass density, f_i and u_i are the body forces and displacements in the i direction. Dots stand for time derivatives and subscripts following a comma mean partial derivatives in space with respect to the x_j coordinate. For an elastic isotropic solid, the Cauchy stress tensor may be expressed in terms of displacements as seen in (2), where λ and μ are the Lamé parameters determining the stiffness properties of the material.

$$\sigma_{ij,j} + f_i = \rho \ddot{u}_j \tag{1}$$

$$\sigma_{ij} = \lambda u_{k,k} \delta_{ij} + \mu (u_{i,j} + u_{j,i}) \tag{2}$$

Though there exists an abundance of numerical methods to address this problem, earthquake simulations at scale have been dominated by finite differences (FD) and finite element (FE) techniques [9]. The formulations for solving (1) and (2) using FD and FE date back to the late 1960s and early 1970s [2, 14, 15, 30]. However, the field of computational seismology only flourished in the 1990s with the ability to perform three-dimensional (3D) simulations at a regional scale using supercomputers [19,20]. Although FD has been the preferred method because of its ease of implementation [21, 22, 33, 34], researchers have successfully developed alternative approaches using low- and high-order FE [7,8,10,11,16,26–28,35]. Other techniques such as boundary elements, coupled boundary-domain elements, and discrete wave-number methods are limited to moderate-size problems with relatively simple geometry and geological conditions.

In recent years it has been demonstrated that, in terms of the computation, memory, and storage requirements, FE approaches can often be 8x more efficient than FD ones [1, 42]. For the most part, this is so because FDs are usually associated with the construction of regular grids to represent the simulation domain—a condition that results in many more grid points than are necessary for the accurate representation of the elastic waves in the stiffer media. Nonetheless, recent advances in FD techniques also allow for more efficient computations [31]. By contrast, FEs enable the use of unstructured meshes tailored to the local wavelength. Depending on the selection of the type of elements used for the mesh, this may lead to significant numerical and computational advantages. Hercules, the CMU earthquake simulator, employs FEs on an adaptive octree mesh to capitalize on such computational advantages.

At a high-level, numerical FE earthquake simulations comprise three main stages: mesh generation, source generation, and solving. Hercules bundles all three stages in an end-to-end approach to perform 3D earthquake simulations due to kinematic faulting [42,45]. It uses a low-order FE method and has been successfully employed in different regional-scale simulations, such as the *TeraShake* and the *ShakeOut* earthquake scenarios [25, 40, 41]. These simulations required meshes of the order of hundreds of millions elements. The same problems, if solved using FD, would require tens of billion elements. Hercules' accuracy of results has been validated with data [37], and successfully verified against other simulators using FD techniques [9].

Fig. 1 sketches the simulation stages implemented in Hercules. The input to the process consists of a set of simulation parameters and a material model. The input parameters include, among others, the maximum wave frequency ($f_{\max}$), the minimum shear wave velocity ($V_{s_{\min}}$), and the required simulation time and Δt defining the total number of time steps. These parameters determine the size and complexity of a simulation. The lower $V_{s_{\min}}$ and the higher $f_{\max}$, i.e., the shorter the wavelength, the more challenging the problem becomes for a prescribed domain. The material model contains the properties of the ground (density and seismic wave velocities). High-resolution models used in present simulations have sizes in the order of tens to hundreds of Gigabytes. To efficiently access the models at run time, they are stored using *etrees*—an indexed data representation format [43].

The mesh generation stage uses the model to produce a discrete mesh suitable for numerical simulation. A simulation mesh is made up of elements, nodes (or vertices) and edges. The parallel mesh generator in Hercules produces an octree mesh with trilinear cubic elements such as the one shown in Fig. 2 [44]. The

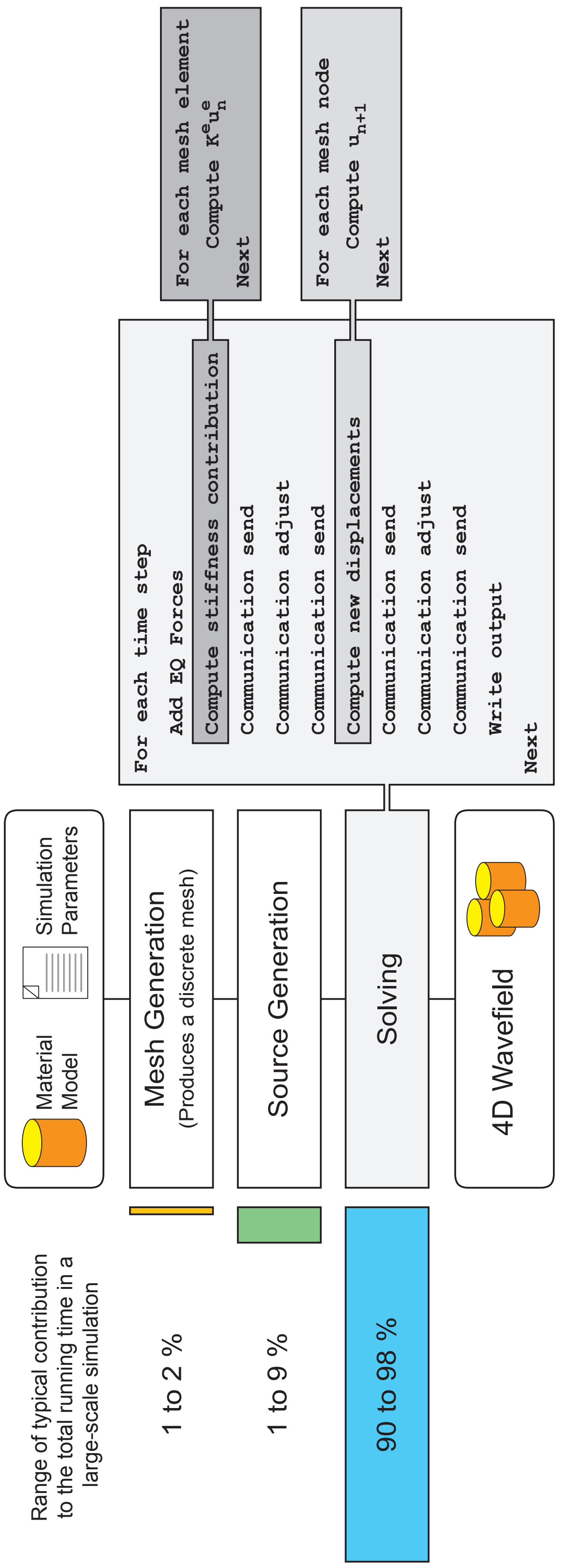


Figure 1: Simulation stages and their typical range of contribution to the total running time along with Hercules' solving algorithm.

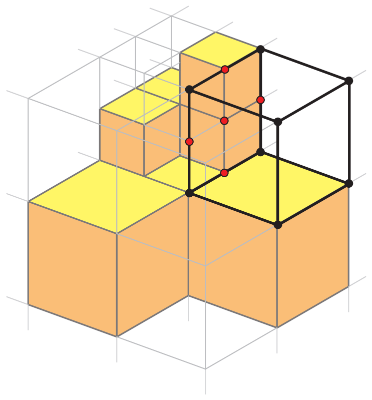


Figure 2: FEM Octree Mesh.

mesh elements are the individual cubes, the corners of the cubes are the mesh nodes. The mesh is partitioned into an equal number of elements across processors and remains in memory for use in the latter stages of the simulation. Mesh generation accounts for 1–2% of the running time in a typical execution of Hercules. Large-scale simulations produce meshes with hundreds of millions to tens of billion elements.

The source generation stage computes the forces produced by the earthquake source (a kinematic representation of a seismic rupture for which the slip along the fault has been prescribed). The values generated during this stage are associated with mesh nodes and correspond to the term f_i in (1). The parallel source generation time makes up for 1–9% of the total running time, depending on the complexity of the earthquake source, the mesh and the location of the source.

The parallel solving stage is the focus of the performance improvements described here. It accounts for the vast majority of the running time (90–98% depending on the simulation scenario). This stage executes the main simulation loop that computes the numerical solution. The main solving loop (shown in Fig. 1) executes for the total number of simulation time-steps. Each iteration of the loop performs the following types of operations: (a) stiffness computation (`Compute stiffness contribution`); (b) communication and waiting, or C+W for short (`Communication send` and `Communication adjust` operations); and (c) other computations (`Compute new displacements` and `Add EQ forces`). As shown later in Section 5, (b) and (c) account only for 10% of Hercules' total solving time. In contrast, (a) accounts for 90% of the solving time, being by large the most computationally demanding process in the execution.

Reducing the computation time spent on the stiffness contribution is the target of the efficient approach described here. Section 3 explains the mathematical formulation for *conventional* stiffness computation and the new *efficient* approach implemented in Hercules.

3 Solution Method and Stiffness Contribution

As previously mentioned, elastic wave propagation problems are governed by the linear momentum equation (1) and the constitutive relation (2). This section deals with the solution of the equations that result from substituting (2) into (1), using a FE approach with two different methods for computing the stiffness contribution.

When applying FE in space to the linear momentum equation using standard Galerkin methods, the combination of (1) and (2) becomes (3). **M** and **K** are the system's mass and stiffness matrices, respectively; **f** is the assembled vector of body forces, which, for the seismic problem, represents the earthquake source; and **u** is the vector of nodal displacements. For convenience, we have omitted terms associated with boundary conditions or intrinsic attenuation, such as viscous damping.

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f} \tag{3}$$

Using central differences to express the second derivative of displacements, $\ddot{\mathbf{u}}$, (3) reduces to a system of difference equations. Furthermore, using a diagonally lumped mass matrix, the system decouples with respect to **M** and the forward solution of displacements for any given node i in the mesh, is given by (4).

$$u^i_{n+1} = (2u^i_n - u^i_{n-1}) + \frac{\Delta t^2}{m^i} f^i_n - \frac{\Delta t^2}{m^i} \left(\sum_{\mathbf{e}} \mathbf{K^e u^e_n} \right)_i \tag{4}$$

Here Δt represents the time step and the subscript n represents a given step at time $t = n\Delta t$. m^i and f^i are the mass and body force associated with the i-th node of interest. $\mathbf{K^e}$ and $\mathbf{u^e}$ are the local stiffness matrix and corresponding vector of displacements of all elements associated with node i.

The last term in (4) implies that a matrix-vector multiplication is performed for all elements at each time step, and the results properly assembled in order to evaluate the stiffness contribution at the i-th node before calculating the new displacement, u^i_{n+1}. Obtaining that last term in (4) corresponds to the `Compute stiffness contribution` method highlighted in the algorithm shown in Fig. 1. As mentioned before, this computation accounts for up to 90% of the total solving time. It is necessary to traverse all the mesh elements assigned to a processor to compute the matrix-vector product $\mathbf{K^e u^e}$ for each element, and assign the values of the resulting vector to the appropriate nodes.

At every time step, once the stiffness contribution has been calculated, the actual solution of (4), that is, the new set of displacements in all nodes, u^i_{n+1}, is done in the `Compute new displacements` method also shown in Fig. 1. This method requires each processor to traverse all of its own mesh nodes at each time step, but in comparison to the stiffness contribution, its computational cost is fairly low (5–10%).

This paper is concerned with the implementation and performance of a new approach for calculating the contribution of the product $\mathbf{K^e u^e}$. Before that, we first review the standard FE approach for computing this term.

3.1 The Conventional Method

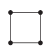
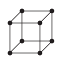
Applying FE and the principle of virtual work to the linear momentum equation, it can be shown that the system's stiffness matrix introduced in (3) is given by (5). The summation means assembling of all the element's individual stiffness matrices, $\mathbf{K^e}$. This assembly is seldom done in practice. Instead, the product $\mathbf{K^e u^e}$ is performed for each element and then stored appropriately. **C** is the tensor of material stiffness, Ω_e is the volume of the element, and ψ' is the matrix of first spatial derivatives of the element's shape functions. The shape functions are expressed in terms of the local coordinates of the element, i.e. $\psi = \psi(\xi_{j=1,2,3})$

$$\mathbf{K} = \sum_{\mathbf{e}} \left(\int_{\Omega_\mathrm{e}} \psi' \mathbf{C} \, (\psi')^T d\Omega_\mathrm{e} \right) = \sum_{\mathbf{e}} (\mathbf{K^e}) \tag{5}$$

Since a lumped mass matrix is used in (3), we say the system is uncoupled with respect to **M**. This allows us to calculate the contribution of each element individually. From this and (5) follows that the stiffness contribution of any given element is given by (6). We refer to this equation as the *Conventional Method* to compute the stiffness contribution given by the product $\mathbf{K^e u^e}$.

$$\mathbf{K^e u^e} = \int_{\Omega_\mathrm{e}} \psi' \mathbf{C} \, (\psi')^T \, d\Omega_\mathrm{e} \mathbf{u^e} \tag{6}$$

Table 1: Number of operations for three different type of elements

				
		1D	2D	3D
		Quadratic	Bilinear	Trilinear
	Conventional	12	64	576
Multiply	Efficient	8	26	56
	(Reduction)	(33%)	(59%)	(90%)
	Conventional	6	56	552
Add	Efficient	6	22	317
	(Reduction)	(0%)	(60%)	(42%)
	Conventional	18	120	1128
Total Ops.	Efficient	14	48	373
	(Reduction)	(22%)	(60%)	(67%)

A great advantage in Hercules comes from the fact that, since all the elements are cubes, a generic set of matrices $\mathbf{K^e}$ associated with the Lamé parameters is calculated only once at the beginning of the simulation, and proportionally scaled using the properties and dimensions of each element at every time step to obtain the product $\mathbf{K^e u^e}$. Thus the operations at each time step in `compute stiffness contribution` are only those of the actual stiffness matrix-displacement vector multiplication.

3.2 The Efficient Method

Based on the work originally introduced in [6], and also presented in [31], we now describe the alternative approach we have implemented in Hercules for computing the stiffness contribution provided by the product $\mathbf{K^e u^e}$.

Let us write the element's shape functions ψ as the product of an auxiliary matrix $\mathbf{A}$ and a vector ϕ as shown in (7). $\mathbf{A}$ is a matrix of constants, $\mathbf{a_{ij}} \in \Re$, and ϕ a matrix composed of terms of the form ξ_j^m, where ξ_j are the local coordinate variables of the master element in the shape functions and $m = 0, 1, ..., k$.

$$\psi = \mathbf{A}^T \phi \tag{7}$$

Then it follows that (6) becomes (8). We will refer to (8) as the *Efficient Method* for calculating the stiffness contribution.

$$\mathbf{K^e u}^e = \mathbf{A}^T \int_{\Omega_e} \phi' \mathbf{C} \, (\phi')^T \, d\Omega_e \, \mathbf{A u}^e = \mathbf{A}^T \mathbf{B A u} \tag{8}$$

At first, it appears that the change of variables introduced in (7) results in a larger number of operations because we now need to compute three matrix-vector multiplications instead of one. However, $\mathbf{B}$ is a sparse matrix. Thus, the product $\mathbf{A}^T\mathbf{BAu}$ can be easily written in expanded form to express the stiffness contribution, which results in considerably fewer total operations than with the conventional method.

A step by step comparison between the conventional and efficient methods for a longitudinal wave propagation problem modeled with a 1D second-order element is included in the Appendix. It illustrates the differences between the two methods. The implementation of the efficient method in Hercules, described in the in Section 4, follows exactly the same scheme shown in the appendix, only that it is done for the 3D case of an 8-node regular hexahedral element (Fig. 2).

Table 1 shows a comparison of the number of operations required using both methods for three different kind of elements in 1D, 2D, and 3D. It also includes the percentage of reduction in the efficient method with respect to the conventional one for each case broken in multiply, add, and total number of operations. The 1D quadratic element reduction is explained in detail in the Appendix. The 2D 4-node square element in the middle of Table 1 is equivalent to the case presented by the authors who originally proposed the efficient method [6]. Our results are consistent with theirs.

The rightmost column in Table 1 shows the results for an 8-node trilinear cubic element. These are of particular interest to us because this is the type of element used in Hercules. Note that the reduction in multiply operations is of one order of magnitude. The reduction in the total number of additions is over 40%. As a result, the implementation of the efficient method in Hercules means a reduction of 67% in the total number of operations required to obtain the stiffness contribution to the solution of the equation of motion with respect to the previous conventional implementation.

When the efficient method was first introduced, it was only presented in detail for a 2D 4-node quadrilateral [6]. Expressions (7) and (8) are in general 3D form. We have expanded the efficient method for the particular case of a 3D 8-node cubic element with excellent results, as seen from Table 1. A detailed evaluation of the performance of the conventional and the efficient methods as well as of their relative differences appears later, in Section 5. We believe that the efficient method may deliver even larger reductions for higher-order 3D elements such as 20-node or 27-node cubes. The reductions, however, would not be as dramatic for non-prismatic elements.

4 Implementation

4.1 Conventional Method

As we mentioned before, in Hercules, since all the elements are cubes, only a set of generic stiffness matrices needs to be computed, and later scaled for the particular properties of each element (the Lamé parameters and size of the element). In the case of a homogeneous 3D element, the construction of the stiffness matrix $\mathbf{K^e}$ in (5), depends on the expansion of the material tensor $\mathbf{C}$. For an elastic isotropic material, $\mathbf{C}$ can be expressed in terms of the Lamé parameters λ and μ, which in Hercules are derived from the density and the seismic velocities stored in the material model and stored for each mesh-element. This implies that the actual stiffness matrix of an element consists of two matrices, one depending on λ and the second depending on μ as in (9), where h is the size of the element. Then the $\mathbf{K^e u^e}$ product for the stiffness contribution becomes (10).

$$\mathbf{K^e} = \mu h \mathbf{K}_1^\mathbf{e} + \lambda h \mathbf{K}_2^\mathbf{e} \tag{9}$$

$$\mathbf{K^e u^e} = \left(\mu h \mathbf{K}_1^\mathbf{e} + \lambda h \mathbf{K}_2^\mathbf{e}\right) \mathbf{u^e} \tag{10}$$

$$c_1 = \mu h \qquad c_2 = \lambda h \tag{11}$$

Since λ, μ, and h are particular to each element, (10) has to be performed at each time step. For this, Hercules defines two constants c_1 and c_2 as defined in (11), that are stored for each element in memory. And the actual computation of the stiffness contribution for the conventional method follows the order denoted by the parentheses in (12)

$$\mathbf{K^e u^e} = \left(c_1 \left(\mathbf{K}_1^\mathbf{e} \mathbf{u^e}\right)\right) + \left(c_2 \left(\mathbf{K}_2^\mathbf{e} \mathbf{u^e}\right)\right) \tag{12}$$

Therefore, computing the stiffness contribution for a given element requires two constant×(matrix×vector) products. For the 8-node cubic elements used in Hercules, each matrix $\mathbf{K}_i^\mathbf{e}$ is of size 24×24 (8 nodes, 3 components).

4.2 Efficient Method

In the efficient method, in accordance with (8), (12) becomes (13). However, we do not perform any of the matrix-vector multiplications implied in (13), but rather explicitly lay out all the necessary operations in the code. That is, we hard-coded the steps shown in (14) for the two components of $\mathbf{K^e}$ associated with c_1 and c_2.

$$\mathbf{K^e u^e} = c_1 \left(\mathbf{A}^T \left(\mathbf{B} \left(\mathbf{A u^e}\right)\right)\right) + c_2 \left(\mathbf{A}^T \left(\mathbf{B} \left(\mathbf{A u^e}\right)\right)\right) \tag{13}$$

$$\begin{aligned} &1. & \alpha_\mathrm{i} &= \sum \mathbf{a}_\mathrm{ij} \mathbf{u}_\mathrm{j} \\ &2. & \beta_\mathrm{i} &= \sum \mathbf{b}_\mathrm{ij} \alpha_\mathrm{j} \quad \text{only if} \quad \mathbf{b}_{ij} \neq 0 \\ &3. & \gamma_\mathrm{i} &= \sum \mathbf{a}_\mathrm{ji} \beta_\mathrm{j} \\ &4. & \left(\mathbf{K^e_m u^e}\right)_i &= c_\mathrm{m} \gamma_\mathrm{i} \qquad \text{where} \quad \mathrm{m} = 1,2 \end{aligned} \tag{14}$$

Here, α_i, β_i, and γ_i are vectors with $\mathrm{i} = 1..24$; and all summations are done for $\mathrm{j} = 1..24$.

As mentioned before, thanks to the fact that many of the elements in matrix $\mathbf{B} = [\mathbf{b}_\mathrm{ij}]$ are zero, the number of multiplications in steps 2 and 3 is significantly reduced. In addition, all the elements in $\mathbf{A} = [\mathbf{a}_\mathrm{ij}]$ are either 1 or -1; thus, once written in the code, steps 1 and 3 entail only additions.

5 Evaluation

The goal of this evaluation is to determine the performance benefits realized by the implementation of the efficient method in Hercules in terms of its running time. In particular, we want to answer the following questions: (1) what is the running time speedup relative to the conventional method? and (2) what effect does the efficient method have in the scalability of the numerical simulation? To answer these questions, we measure the execution time of simulations using both the efficient and conventional approaches under different resource and problem size configurations, which allows us to determine their performance and scalability. The experimental setup is as follows.

5.1 Study Case: Chino Hills Earthquake

The Chino Hills earthquake (Mw 5.4) of July 29, 2008 was the strongest earthquake in the greater Los Angeles metropolitan area since the Northridge earthquake in 1994 [23]. Because the ground motion it generated was recorded by several seismic networks in southern California, its occurrence constituted an excellent opportunity to test the capabilities of different earthquake simulators.

We presented a preliminary study on validation of results using Hercules to compare data with synthetics from a simulation with $f_\mathrm{max} = 2$ Hz and $V_{\mathrm{s_{min}}} = 200$ m/s [37]. Fig. 3 shows a snapshot of the simulation we did for Chino Hills earthquake. The modeling domain is a box with dimensions: 135 km $\times$ 180 km $\times$ 62 km, which includes the greater Los Angeles basin. This simulation required an unstructured mesh with only 630 million elements. An equivalent simulation using FD regular grids would require over 20 billion elements. Our simulation recreated 100s of shaking using 100,000 time steps and consumed 72,000 supercomputing service units (SUs) (less than 18 hours, in just 4K processor cores).

Here, we chose to use this same scenario to evaluate the performance and scalability of the efficient method implementation vis-à-vis the conventional method. In simulations used for science goals, many of the input parameters are varied to better understand the wave propagation phenomena. Typical simulation times are of the order of 200–300 seconds for strong motion earthquakes, which may require a number of simulation time steps in the range of 40,000 to 200,000. However, for convenience, all simulations

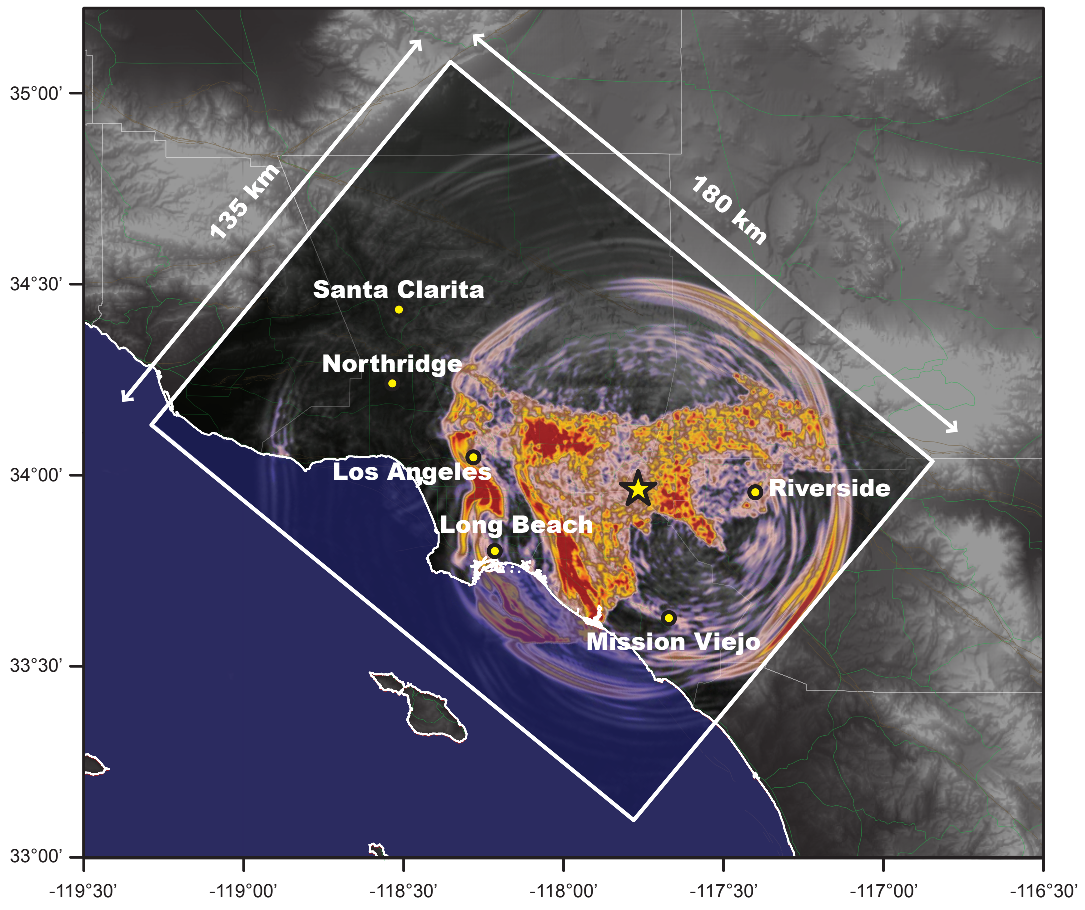


Figure 3: 2008 Chino Hills earthquake and modeling domain used as testbed, including a snapshot of the ground response for the horizontal surface velocity during a simulation with $f_{\max} = 2$ Hz and $V_{s_{\min}} = 200$ m/s.

performed for this study were run only for 2,000 time steps, since the main objective of the present study was to examine the performance of the two implementations. The simulation rate for the solving step is very stable; thus, shorter simulations are representative of full-length simulations. Short simulations allow good coverage in the resource parameter space.

For the present simulations, the input material model, an *etree* database of 74 Gigabytes remained the same, and the minimum shear wave velocity $V_{s_{\min}}$ was kept fixed at 500 m/s. The remaining parameters ($f_{\max}$ and Δt) were varied to change the problem size (amount of work) for different resource configurations as described ahead. The meshes are highly unstructured and have large variations in element sizes. The size of the mesh, measured in the total number of elements, indicates the problem size in the experiments described below.

5.2 Resource and Problem Sizes

We varied the number of available resources (CPU cores) and the problem size as follows: (1) Fixed-size resource (vertical scaling); (2) Fixed problem size (strong scaling); and (3) Isogranular problem size (weak scaling) [36]. Our experiments were carried out in the Teragrid Kraken Supercomputer operated by the

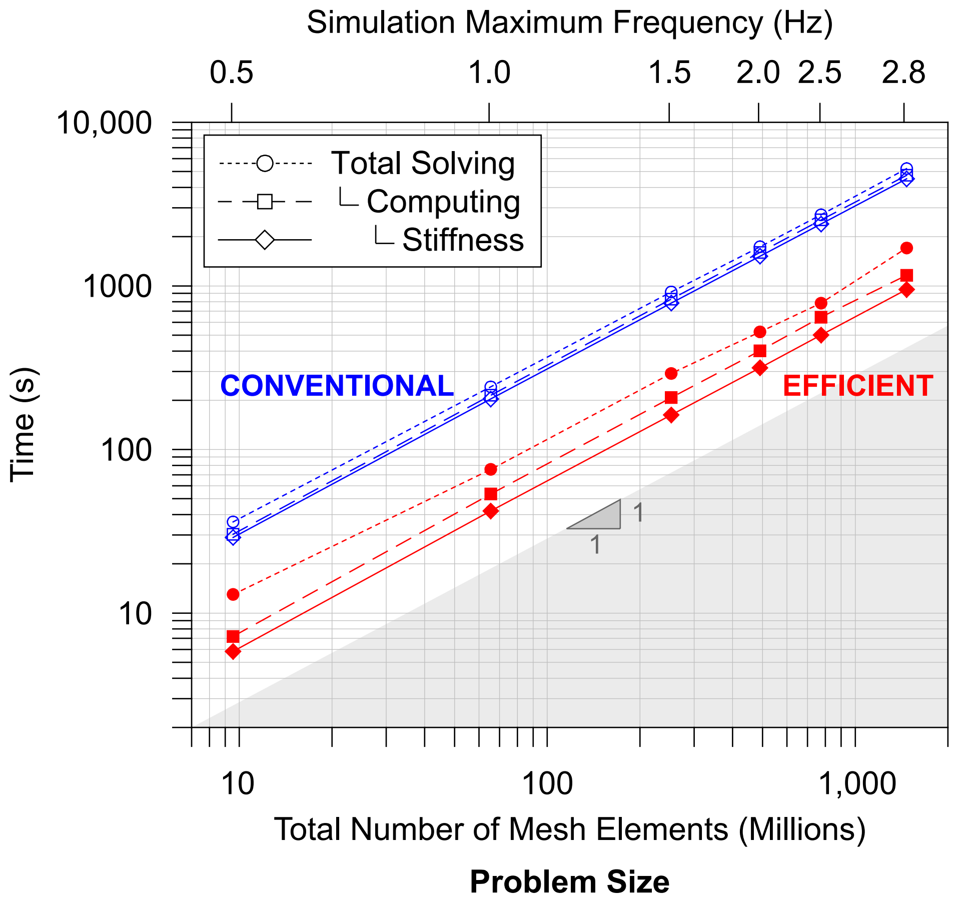


Figure 4: Scaling for the fixed-size resource case. The problem size varies across executions (X axis), while the number of CPU cores is fixed at 1032. The Y axis shows the elapsed wall-clock time.

National Institute for Computational Sciences (NICS) [32]. The system has a total of 99,072 compute cores and 129 TB of aggregate memory. Each compute node has 16 GB of RAM and 12 CPU cores in two 2.6 GHz six-core AMD Opteron processors (Istanbul). The nodes run the Cray Linux Environment and are connected via Cray SeaStar2+ routers.

5.3 Performance with Fixed Resource Size

In this set of experiments the size of the available resources for the computation is fixed at 1032 CPU cores. The problem size varied from 1 million to 1.47 billion mesh elements. The objective of these experiments is to measure the benefit of the optimization introduced by the efficient method at different problem sizes and validate that the efficient approach still exhibits the expected linear running-time behavior. The results for these experiments are shown in the log-log plot in Fig. 4. The X axis is the problem size given in number of mesh elements in $\log_{10}$ scale. The Y axis is the elapsed wall-clock time in seconds for the simulation solving-phase, also in $\log_{10}$ scale; lower means faster execution. The corresponding simulation maximum frequency for each problem size is displayed in the X2 axis at the top of the graph. The graph has two groups of lines, the top group (hollow symbols ○□◇) represents the running time of the conventional method, the

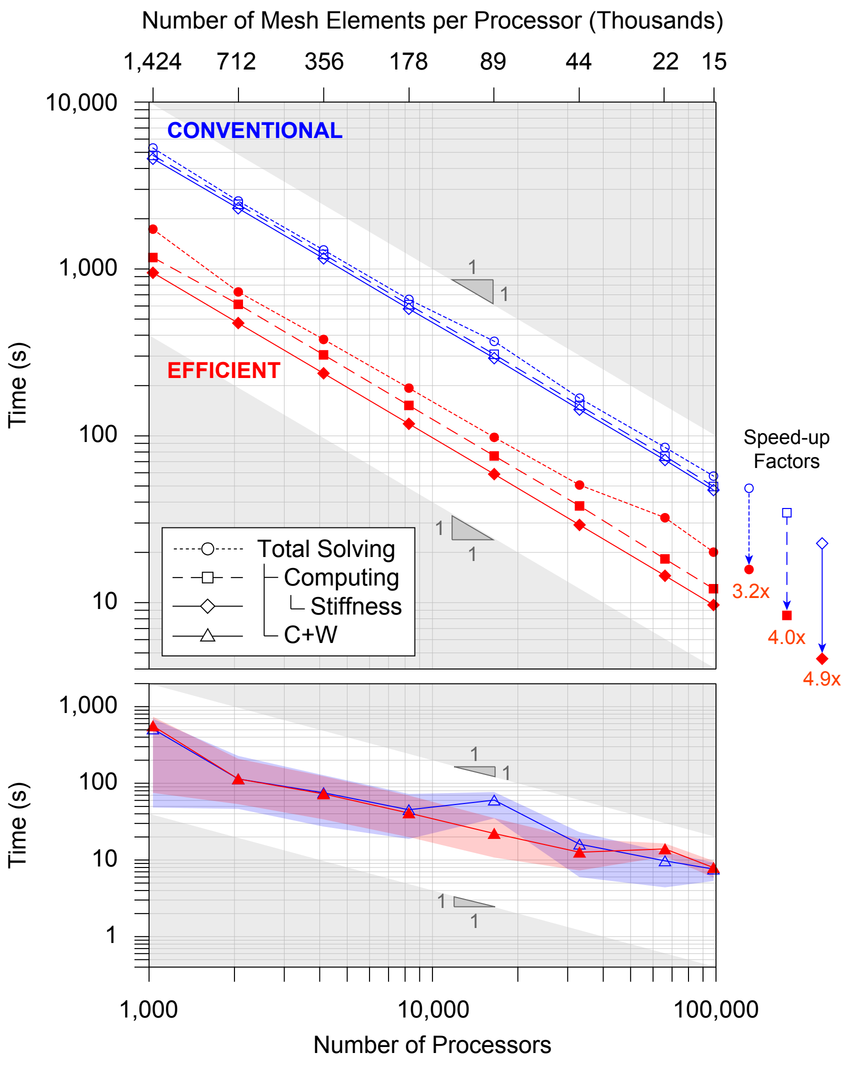


Figure 5: Fixed problem size scalability.

bottom group (solid symbols ●■♦) corresponds to the efficient method. In each group there are three lines, top dotted line (circles) is the total solving time. The middle dashed line in each group (squares) is the time spent on computation. The bottom solid line (diamonds) is time spent on the stiffness computation. This convention is followed across all experiments. The difference between the top and middle lines corresponds to the communication time due to either sending data to or waiting for data from other processors (C+W). In both approaches, the elapsed time is linear with respect to the problem size. A line with a slope of 1 (shaded area) is included in the figure for reference. The efficient method results in a 3x speedup on average across the different problem sizes.

5.4 Fixed Problem Size Scalability:

For these experiments, the problem size remains fixed and the number of processors varies. This is commonly known as strong scaling. The problem size corresponds to an earthquake simulation with a maximum wave frequency of 2.8 Hz and a mesh with 1.5 billion elements. The results are shown in Fig. 5. This is a log-log plot with the number of processors in the X axis and the elapsed time in the Y axis. The top part of the graph displays the elapsed time for the numerical solving phase for both the conventional and efficient approaches. Both approaches exhibit linear scaling. For reference, a clear band with slope -1 is shown in the figure. The efficient approach exhibits speedups of 4.9x for the stiffness computation time (ratio of the hollow diamond data points $\diamond$ to the solid diamond points ⧫). When all the other per-time step computations are taken into consideration, the resulting speedup is 4x (□ to ■). Once the communication and other operations are taken into account, the resulting overall speedup for the solving time is 3.2x on average (ratio of $\circ$ to $\bullet$). The speedups relative to the conventional approach are shown in the upper half of Fig. 6.

The total communication time decreases inversely proportional to the number of processors, but the slope is < -1. The bottom plot in Fig 5 shows this effect. The X axis is the number of processors and the Y axis is the C+W time. The shaded region around the lines shows the accumulated minimum and maximum waiting times experienced by different processing elements. Notice that the C+W time was very similar for both procedures.

However, while C+W accounted only for approximately 10% of the time in the conventional method, in the efficient approach this time grows to 30% of the total solving time. As expected, once the cost of the main inner loop computation is dramatically reduced for the efficient approach, then other per-time step operations start having a larger effect in the overall performance [3]. This effect is displayed in the upper half of Fig. 6, where the X axis is the number of processors used in the experiment, and the Y axis is the relative time (%) spent in the three main operations performed in the solving phase. The lines show the speedup of the efficient method vs. that of the conventional one (Y2 axis). For each pair of bars: the bar on the left shows the breakdown for the conventional approach (normalized to the solving time of the conventional method); and the bar on the right shows the breakdown for the efficient method (normalized to the time of the conventional method as well). In each bar the bottom region is the portion spent on `compute new displacement`, the middle region is the portion spent on `compute stiffness contribution` and the top region is C+W. As it is evident, once the stiffness computation is accelerated, then the other computations and C+W have a larger relative contribution to the running time. What previously were minor load imbalances in the displacement computation and in C+W, now start having an impact in the achievable performance. This is particularly apparent in the 66K and 99K processors cases. This does not mean that the efficient code is not scalable for a number of processors greater than 66K. On the contrary, it means that the efficient approach can handle much larger problem sizes as it is shown in the next set of experiments. However, this behavior also indicates that any improvement that can be achieved in the C+W and other ancillary tasks will have a greater impact on the overall performance of the efficient method than it does for the conventional method. Conversely, with the speedup now gained in the computation of the stiffness-contribution step, it becomes more important to reduce the time spent in C+W and other auxiliary processes.

5.5 Isogranular Scalability

In these experiments, the amount of work per processor is approximately constant across simulations and the size of the problems or total amount of work grows as the number of processors increases. We varied the number of processors from 1000 to 99K. Fig. 7 shows the elapsed time (Y-axis, log scale) vs. the number of processors (X axis, log scale). The mesh size varies from 120 million to 11.6 billion elements. The mesh size and maximum simulation frequency are included in the X2 axis at the top of the plot. The lower graph shows the cumulative C+W time, which stays relatively constant at different scales, since the C+W time is a function of both the amount of data exchanged among processors and additional computation performed in each processor. The relative speedups and time breakdown for these experiments are shown in the lower

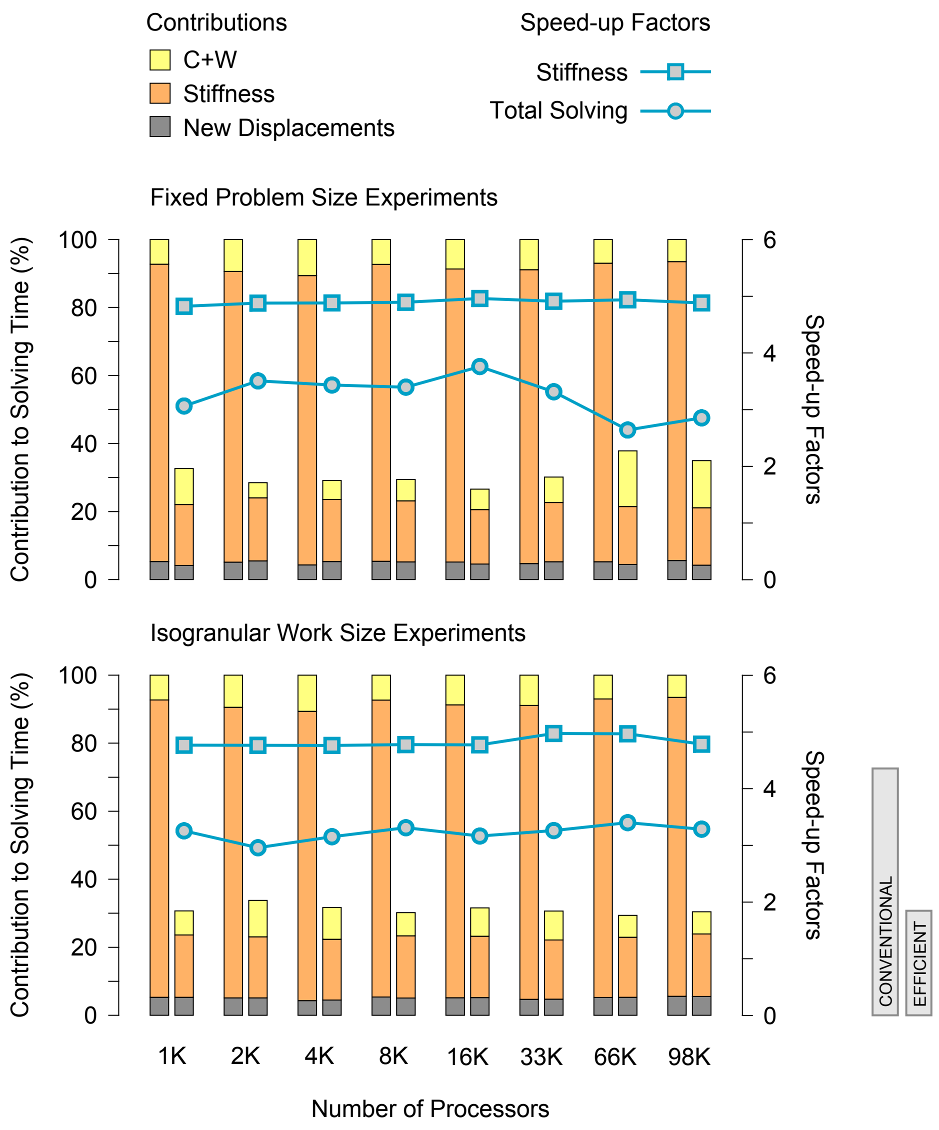


Figure 6: Contribution of each module to the total solving time of both methods, using as reference the results for the conventional method, for the fixed problem size (top) and isogranualar (bottom) experiments. Speed-up factors for the total solving and stiffness computation are superimposed.

half of Fig. 6. The efficient method exhibits good isogranular scalability and achieves speedup factors of 3.2x for the total solving time when compared to the conventional approach.

Overall, the evaluation done for the different experiments shows that the net impact of the efficient method is that the simulation phase has a speedup of at least 3x, and even faster in some cases. By reducing the solving time, the straight-forward immediate implication is that now we will be able to run simulations that are 3x larger in the same elapsed time or the same simulation 3x faster. For instance, based on the results presented here, we can estimate that the 2 Hz simulation of the 2008 Chino Hills earthquake we carried out previously with a $V_{s_{\min}}$ of 200 m/s for 100 s duration, would take only 15 minutes to complete on 99K core processors.

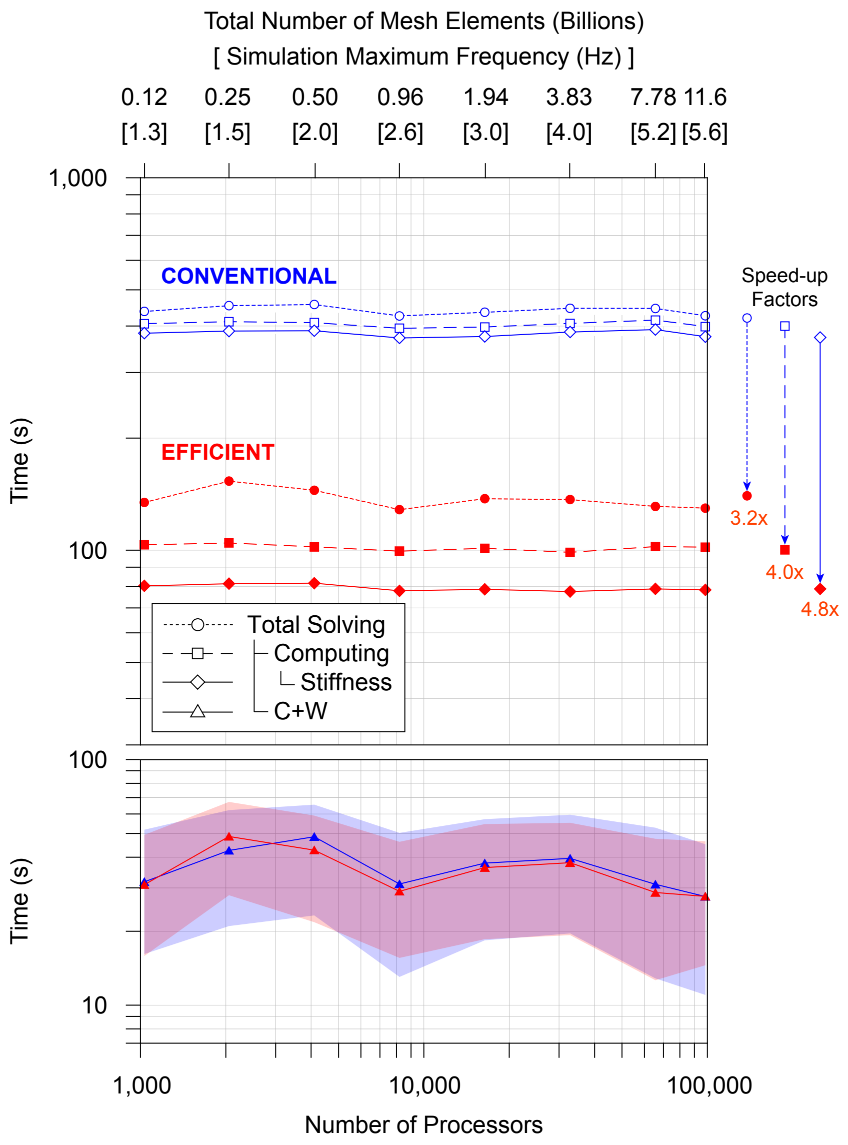


Figure 7: Isogranular Scalability.

6 Discussion and Future Work

The efficient approach greatly shortens the overall simulation time by dramatically reducing the required number of compute cycles. The regained compute cycles can be utilized to increase the realism of our simulations by (i) increasing the maximum wave frequency, $f_{\max}$, and reducing the minimum shear wave velocity, $V_{s_{\min}}$, i.e., refining the mesh; and (ii) incorporating more complex physical processes such as nonlinear soil and site effects [38, 39] or problems of urban seismology [17, 18]. A level of complexity that places high computational demands even for today's capability machines.

As part of addressing this challenge we are currently working on further reducing the required simulation time. As shown in the evaluation, although the C+W time remains roughly the same for both approaches, its relative contribution to the solving time becomes much larger in the efficient method. We are developing an alternate communication approach that promises to reduce the C+W time for both methods. The basic ideas are: (1) reducing the number of communication rounds in each time step; and (2) enabling alternative load balancing schemes.

In the current communication approach, Hercules performs four rounds of communication per time step [44]. These rounds of communication correspond to the `Communication send` calls in the solving loop (Fig. 1). These data exchanges are interleaved with the displacement computation (`Compute new displacement`) and two small computation steps (`Communication adjust`). In Hercules, the computation is load balanced with respect to the number of mesh elements. All PEs have the same number of elements. However, the displacement computation and adjust computations are not fully balanced. Although the adjust computation is fairly small, some PEs perform more work than others in this step, and there are PEs that perform no work at all during the adjust computation. This load imbalance accounts for the majority of the C+W time.

To address these issues, we are implementing a new mechanism that replaces the four communication rounds with a single data exchange. In general, this modification results in a PE communicating with the same number of neighboring PEs with minimal increase in the message sizes. By having a single data exchange per solving time step, we can then implement a new load balancing scheme that takes into account the computation performed in other processes. PEs that perform very little work in the adjust phase can process more elements that those PEs that perform more work in the adjust computation. Rebalancing the load in this manner with the current communication mechanism is not desirable because it would increase the wait time due to embedded synchronization points. In the new communication scheme, all adjust computations can be performed at once and then, the data exchange occurs. The new displacement and second adjust computations can proceed right after a PE has received the needed data and can continue to the next iteration.

A different set of optimizations that we are currently developing are aimed at reducing the I/O time. Any benefits obtained from speeding up the computation may be completely negated by the time spent saving the output and intermediate checkpoints. We are developing new mechanisms that use a fraction of compute cycles to compress the 4D wavefield output using a combination of domain specific approaches and floating point encoding [29]. The major objective is to reduce the required I/O bandwidth and storage space and, at the same time, making it easier to post-process and analyze the output wavefield by using a data-intensive approach.

7 Related Work

Reducing the amount of work needed to obtain a result has been one of the main principles used to speed up computation and vastly studied in the field of Computational Complexity [5]. In Computational Sciences and High-End Computing, this approach is often embodied in the reduction of the number of floating point operation. Canonical examples include efficient algorithms to operate on large sparse matrices, such as those implemented in LAPACK and ScaLAPACK [4, 12, 13]. The choice of method to solve the partial differential equations (PDEs) has a major impact in the number of operations and runtime memory requirements. As discussed in Section 2, efficient FD and FE approaches yield major speedups over traditional FD implementations [31, 42]. Hercules implements a state-of-the-art FE approach with adaptive mesh refinement that is often 8x faster, and requires in the order of 1/8 of the memory, when compared to contemporary FD approaches for seismic simulation. Balazovjech et al. proposed an efficient method for modeling seismic wave propagation for the 1D and 2D cases [6]. The work presented here implements an efficient method for the 3D case and is applied to large-scale regional seismic simulations. Traditional code optimizations, such as loop unrolling and data layout reordering have recently been applied to other seismic simulators and reported to yield a 30% improvement of the running time [24]. In Hercules, these optimizations are left to the compiler since hand coded optimizations are often less portable and hard to maintain. The 3x speedups in Hercules result from a reduction in the number of floating point operations.

8 Conclusion

In this study of earthquake wave propagation we have implemented on Hercules an efficient method for incorporating the stiffness matrix-displacement vector product—an operation that normally consumes the bulk of the total solving time at each time step (up to 90%). A comparative analysis of the performance of the efficient method with respect to that of the conventional method showed that the efficient method resulted in a speedup of about 4.8x in the stiffness-displacement computation; 4x in computing time; and 3.2x in total solving time (including communication). On the other hand, both methods exhibit linear scalability under different conditions: fixed resource size (vertical scaling); fixed problem size (strong scaling); and isogranular problem size (weak scaling).

By reducing the solving time, the immediate implication is that it is now possible to run simulations on Hercules that are 3x larger in the same elapsed time or the same simulation 3x faster. The real main benefit of the combined linear scalability of our approach and of the speedup enabled by the efficient methodology is that in the future we will be able to carry out more realistic simulations with higher resolution and incorporating more complex physical problems such as nonlinear soil and site effects, and problems of urban seismology, which require much more computation. It will also allow us to simulate earthquakes in almost real time, following a destructive event. This is of great importance for emergency response, recovery, and reconstruction activities.

A The Conventional vs. Efficient Methods: An illustration on a 1D problem

In Sections 3 and 4 we presented the basic formulation of both the conventional and efficient methods and their implementation in Hercules. It might, however, not be apparent to readily visualize the mathematical steps in place and the contrast that exists between the two methods. Here we illustrate the core difference between them by means of a simplified problem.

For that we will evaluate the number of operations required to obtain the product $\mathbf{K^e u^e}$ for a longitudinal wave propagation problem, modeling an elastic bar in free vibration with a 1D second-order element. In this case, the linear momentum equation shown in Section 2 reduces to (15). σ is the normal stress, ρ the mass density, and u the displacement, which is both a function of time and space, i.e., $u = u(x,t)$. Here, spatial derivatives will be denoted with a prime. Dots stand for derivatives with respect to time.

$$\sigma' = \rho \ddot{u} \tag{15}$$

The internal virtual work in a given element for a homogeneous material with Young's modulus, E, constant, can be written as seen in (16).

$$V = \int_{\Omega} \upsilon' \sigma d\Omega = E \int_0^h \upsilon' u' dx \tag{16}$$

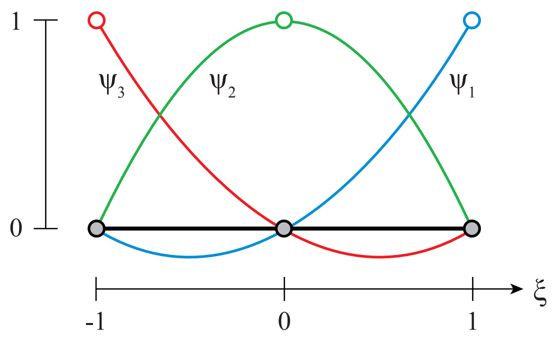


Figure 8: 1D element with Lagrangian quadratic shape functions $\psi_i(\xi)$.

Here, $\sigma = E\varepsilon$ and $\varepsilon = u'$, therefore $\sigma = Eu'$. With this, (15) is the one-dimensional wave equation. In (16), υ is a test function that depends only on x, i.e., $\upsilon = \upsilon(x)$; and h is the mesh size, or element's length. Introducing the change of coordinates in (17), (16) becomes (18).

$$x = \frac{h}{2}(\xi + 1) \tag{17}$$

$$V = \frac{2E}{h}\int_{-1}^{1} \upsilon'(\xi)u'(\xi,t)dx \tag{18}$$

The displacement and test functions are approximated by (19), where ψ are the shape functions of the element in local ξ coordinates. Then, the internal virtual work (18) becomes (20).

$$\begin{aligned} u(\xi,t) &= \psi^T(\xi)\mathbf{u}^\mathbf{e}(t) \\ \upsilon(\xi) &= \psi^T(\xi)\mathbf{v} \end{aligned} \tag{19}$$

$$\begin{aligned} V &= \mathbf{v}^T \frac{2E}{h}\int_{-1}^{1} \psi'\psi'^T d\xi \, \mathbf{u}^\mathbf{e} \\ &= \mathbf{v}^T\mathbf{K}^\mathbf{e}\mathbf{u}^\mathbf{e} \end{aligned} \tag{20}$$

It follows from (20) that the element stiffness is given by (21). This is a particular case of (6), seen in Section 3, in which $\mathbf{C} = 2E/h$ is constant, thus can be taken out of the integrand.

$$\mathbf{K} = \frac{E}{2h}\int_{-1}^{1} \psi'\psi'^T d\xi \tag{21}$$

$$\psi^T(\xi) = \left\{ \frac{1}{2}\xi - \frac{1}{2}\xi^2 \, , \, 1-\xi^2 \, , \, \frac{1}{2}\xi + \frac{1}{2}\xi^2 \right\} \tag{22}$$

Adopting a 1D element with Lagrangian quadratic shape functions as shown in Fig. 8 and defined by (22), one can easily expand (21) and solve the stiffness contribution given by the product $\mathbf{Ku}$ for the two methods presented in Section 3 as follows.

A.1 Conventional Method

Replacing (22) into (21) and multiplying with the displacement vector we can expand the product **Ku** as seen in (23). Not counting the operations necessary to obtain the outer constant $E/3h$, the matrix-vector product in (23) requires 9 multiplications, then 2 additions per row, and finally 3 more multiplications. This yields a total 18 operations: 12 multiplications and 6 additions.

$$\mathbf{Ku} = \frac{E}{3h}\begin{bmatrix} 7 & 8 & -1 \\ 8 & 16 & -8 \\ -1 & -8 & 7 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} \tag{23}$$

A.2 Efficient Method

To apply the efficient method we need to write (8) in Section 3 using (7) and (21). This yields (24) as the new expression for the 1D element stiffness matrix. And the auxiliary matrix **A** and vector ϕ that satisfy (7) are shown in (25).

$$\mathbf{K} = \frac{2E}{h}\mathbf{A}^T \int_{-1}^{1} \phi' \left(\phi'\right)^T d\xi\, \mathbf{A} \tag{24}$$

$$\mathbf{A}^T\phi = \begin{bmatrix} 0 & \frac{1}{2} & -\frac{1}{2} \\ 1 & 0 & -1 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}\begin{Bmatrix} 1 \\ \xi \\ \xi^2 \end{Bmatrix} = \begin{Bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{Bmatrix} \tag{25}$$

Using **A** and ϕ as in (25), the element stiffness matrix (24) becomes (26), which can be written in compact form as in (27).

$$\mathbf{K^e} = \frac{E}{2h}\begin{bmatrix} 0 & 1 & -1 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & \frac{8}{3} \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 \\ 1 & 0 & -1 \\ -1 & -2 & 1 \end{bmatrix} \tag{26}$$

$$\mathbf{K^e} = \frac{E}{2h}\mathbf{A}^T\mathbf{BA} \tag{27}$$

Finally, to compute the stiffness contribution one must compute the product $\mathbf{K^e u^e}$ grouping the terms in (27) with $\mathbf{u^e}$ in the order denoted by the parenthesis in (28).

$$\mathbf{K^e u^e} = \frac{E}{2h}\left(\mathbf{A}^T\left(\mathbf{B}\left(\mathbf{Au}\right)\right)\right) \tag{28}$$

As when comparing (6) and (8) in Section 3, (26) and (28) seem, at first sight, to entail a larger number of operations than the equivalent **Ku** product in the conventional method (23). However, as previously mentioned, matrix **B** is sparse and made up of constants. Thus, (28) can be easily expanded explicitly, resulting in (29).

$$\alpha = \mathbf{A}\mathbf{u} = \left\{ \begin{array}{c} 2u_2 \\ u_1 - u_3 \\ -u_1 - 2u_2 \end{array} \right\} \quad \beta = \mathbf{B}\alpha = \left\{ \begin{array}{c} 0 \\ 2\alpha_1 \\ \frac{8}{3}\alpha_3 \end{array} \right\}$$
$$\gamma = \mathbf{A}^T\beta = \left\{ \begin{array}{c} \beta_1 - \beta_2 \\ 2(\beta_1 - \beta_3) \\ \beta_2 + \beta_3 \end{array} \right\} \qquad \mathbf{K}\mathbf{u} = \frac{E}{2h} \left\{ \begin{array}{c} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{array} \right\} \tag{29}$$

It follows from (29), that the breakdown of operations in the efficient method to compute the stiffness contribution for a 1D quadratic element is: 8 multiplications and 6 additions; for a total of 14 operations. A net difference of 4 with respect to the conventional method, representing a reduction of 22%.

Table 1 in Section 3, shows that the reduction accomplished with the efficient method following the procedure shown here significantly increases for a 4-node square, and an 8-node trilinear cube. This is because for a trilinear approximation the matrix **B** is even more sparse, and matrix **A** consists only of 1 and -1 terms.

References

[1] V. Akcelik, J. Bielak, G. Biros, I. Ipanomeritakis, A. Fernandez, O. Ghattas, E. Kim, J. López, D. O'Hallaron, T. Tu, and John Urbanic. High resolution forward and inverse earthquake modeling on terascale computers. In *Proceedings of Supercomputing SC'2003*, Phoenix AZ, USA, Nov 2003. ACM, IEEE. Available at `www.cs.cmu.edu/~ejk/sc2003.pdf`.

[2] Z. Alterman and F. C. Karal. Propagation of elastic waves in layered media by finite difference methods. *Bulletin of the Seismological Society of America*, 58(1):367–398, 1968.

[3] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, New York, NY, USA, 1967. ACM.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2nd edition, 1995.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0-521-42426-4.

[6] Martin Balazovjech and Ladislav Halada. Effective computation of restoring force vector in finite element method. *Kybernetika*, 49(6):767–776, 2007.

[7] Hesheng Bao, Jacobo Bielak, Omar Ghattas, Loukas F. Kallivokas, David R. O'Hallaron, Jonathan R. Shewchuk, and Jifeng Xu. Earthquake ground motion modeling on parallel computers. In *SC '96: Proceedings of the 1996 ACM/IEEE Conference on High Performance Networking and Computing*, page 13, Pittsburgh, Pennsylvania, United States, 1996. IEEE Computer Society.

[8] Hesheng Bao, Jacobo Bielak, Omar Ghattas, Loukas F. Kallivokas, David R. O'Hallaron, Jonathan R. Shewchuk, and Jifeng Xu. Large-scale simulation of elastic wave propagation in heterogeneous media

on parallel computers. *Computer Methods in Applied Mechanics and Engineering*, 152(1-2):85–102, 1998.

[9] J. Bielak, R. W. Graves, K. B. Olsen, R. Taborda, L. Ramírez-Guzmán, S. M. Day, G. P. Ely, D. Roten, T. H. Jordan, P. J. Maechling, J. Urbanic, Y. Cui, and G. Juve. The ShakeOut earthquake scenario: Verification of three simulation sets. *Geophysical Journal International*, 180(1):375–404, 2010.

[10] Jacobo Bielak, Omar Ghattas, and Eui Jin Kim. Parallel octree-based finite element method for large-scale earthquake ground motion simulation. *Computer Modeling in Engineering and Sciences*, 10(2):99–112, 2005.

[11] Jacobo Bielak, Jifeng Xu, and Omar Ghattas. Earthquake ground motion and structural response in alluvial valleys. *Journal of Geotechnical and Geoenvironmental Engineering, ASCE*, 125(5):413–423, 1999.

[12] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petiet, K. Stanley, D. Walker, and R. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1997.

[13] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Soft*, 28(2):135–151, 2002.

[14] David M. Boore. Love waves in nonuniform wave guides: Finite difference calculations. *Journal of Geophysical Research*, 75(8):1512–1527, 1970.

[15] David M. Boore. *Methods in Computational Physics*, volume II, chapter Finite difference methods for seismic wave propagation in heterogeneous materials. ed. Bolt, B. A., Academic Press, 1972.

[16] Emmanuel Chaljub, Dimitri Komatitsch, Jean-Pierre Vilotte, Yann Capdeville, Bernard Valette, and Gaetano Festa. Spectral-element analysis in seismology. In Ru-Shan Wu and Valérie Maupin, editors, *Advances in Wave Propagation in Heterogeneous Media*, volume 48 of *Advances in Geophysics*, pages 365–419. Elsevier, 2007.

[17] Antonio Fernández-Ares. *Urban Seismology: Interaction between earthquake ground motion and response of urban regions*. PhD thesis, Carnegie Mellon University, 2003.

[18] Antonio Fernández-Ares and Jacobo Bielak. Urban Seismology: Interaction between earthquake ground motion and multiple buildings in urban regions. In *Proceedings of the 3rd International Symposium on the Effects of Surface Geology on Seismic Motion*, Grenoble, France, August 2006. IASPEI and IAEE.

[19] Arthur Frankel. Three-dimensional simulations of ground motions in the San Bernardino Valley, California, for hypothetical earthquakes on the San Andreas fault. *Bulletin of the Seismological Society of America*, 83(4):1020–1041, 1993.

[20] Arthur Frankel and John Vidale. A three-dimensional simulation of seismic waves in the Santa Clara Valley, California, from a Loma Prieta aftershock. *Bulletin of the Seismological Society of America*, 82(5):2045–2074, 1992.

[21] Takashi Furumura and Kazuki Koketsu. Parallel 3-D simulation of ground motion for the 1995 Kobe earthquake: The component decomposition approach. *Pure and Applied Geophysics*, 157(11-12):2047–2062, 2000.

[22] Robert W. Graves. Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences. *Bulletin of the Seismological Society of America*, 86(4):1091–1106, 1996.

[23] Egill Hauksson, Karen Felzer, Doug Given, Michal Giveon, Susan Hough, Kate Hutton, Hiroo Kanamori, Volkan Sevilgen, Shengji Wei, and Alan Yong. Preliminary Report on the 29 July 2008 Mw 5.4 Chino Hills, Eastern Los Angeles Basin, California, Earthquake Sequence. *Seismological Research Letters*, 79(6):855–866, 2008.

[24] Nguyen HT, Cui Y, Olsen KB, and Lee K. Single cpu optimizations of scec awp-olsen application (abstract). In *Proc. SCEC annual meeting*, 2009.

[25] Lucile M. Jones, Richard Bernknopf, Dale Cox, James Goltz, Kenneth Hudnut, Dennis Mileti, Suzanne Perry, Daniel Ponti, Keith Porter, Michael Reichle, Hope Seligson, Kimberley Shoaf, Jerry Treiman, and Anne Wein. The ShakeOut scenario. Technical Report USGS-R1150, CGS-P25, U.S. Geological Survey and California Geological Survey, 2008.

[26] Martin Käser and Michael Dumbser. An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshes — I. The two-dimensional isotropic case with external source terms. *Geophysical Journal International*, 166(2):855–877, 2006.

[27] Dimitri Komatitsch, Christophe Barnes, and Jeroen Tromp. Simulation of anisotropic wave propagation based upon a spectral element method. *Geophysics*, 65(4):1251–1260, 2000.

[28] Dimitri Komatitsch and Jean-Pierre Vilotte. The spectral element method: An efficient tool to simulate the seismic response of 2D and 3D geological structures. *Bulletin of the Seismological Society of America*, 88(2):368–392, 1998.

[29] Julio López, Leonardo Ramirez, Jacobo Bielak, and David O'Hallaron. BEMC: A searchable, compressed representation for seismic wavefields. In *22nd Int. Conf on Scientific and Statistical Database Management (SSDBM'10)*, 2010.

[30] J. Lysmer and L. A. Drake. A finite element method for seismology. In B. Alder, S. Fernbach, and B.A. Bolt, editors, *Methods in Computational Physics*, volume 11, chapter 6. Academic Press, New York, 1972.

[31] Peter Moczo, Jozef Kristek, M. Galis, P. Pazk, and M. Balazovjech. The finite-difference and finite-element modeling of seismic wave propagation and earthquake motion. *Acta Physica Slovaca*, 57(2):177–406, 2007.

[32] National Institute for Computational Sciences (NICS): University of Tennessee & Oak Ridge National Laboratory. Teragrid kraken supercomputer. `http://www.nics.tennessee.edu/computing-resources/kraken`.

[33] Kim B. Olsen, Ralph J. Archuleta, and Joseph R. Matarese. Three-dimensional simulation of a magnitude 7.75 earthquake on the San Andreas fault. *Science*, 270(5242):1628–1632, December 1995.

[34] Arben Pitarka, Kojiro Irikura, Tomotaka Iwata, and Haruko Sekiguchi. Three-dimensional simulation of the near-fault ground motion for the 1995 Hyogo-Ken Nanbu (Kobe), Japan, earthquake. *Bulletin of the Seismological Society of America*, 88(2):428–440, 1998.

[35] Geza Seriani and Enrico Priolo. Spectral element method for acoustic wave simulation in heterogeneous media. *Finite Elements in Analysis and Design*, 16(3–4):337–348, 1994.

[36] Xian-He Sun and Lionel M. Ni. Scalable problems and memory-bounded speedup. *J. Parallel Distrib. Comput.*, 19(1):27–37, 1993.

[37] R. Taborda, H. Karaoglu, J. Bielak, J. Urbanic, Julio López, and L. Ramírez-Guzmán. Chino Hills — A highly computationally efficient 2 Hz validation exercise. In *Proceedings and Abstracts of the 2009 SCEC Annual Meeting*, Palm Springs, CA, September 13-16, 2009.

[38] Ricardo Taborda and Jacobo Bielak. Three-dimensional modeling of earthquake ground motion in basins, including nonlinear wave propagation in soils. Final Technical Report 08HQGR0018, USGS, 2008.

[39] Ricardo Taborda and Jacobp Bielak. Three-dimensional modeling of earthquake ground motion including nonlinear wave propagation in soils. In *Abstracts of the 2009 SSA Annual Meeting*, Monterey, CA, April 8–10, 2009.

[40] Ricardo Taborda, Leonardo Ramírez-Guzmán, Julio López, John Urbanic, Jacobo Bielak, and David O'Hallaron. ShakeOut and its effects in Los Angeles and Oxnard areas. *Eos Transcripts of the American Geophysical Union*, 88(52): Fall Meeting Supplement, Abstract IN21B–0477, December 2007.

[41] Ricardo Taborda, Leonardo Ramírez-Guzmán, Tiankai Tu, E. J. Kim, Julio López, Jacobo Bielak, Omar Ghattas, and David O'Hallaron. Scaling up TeraShake: A 1-Hz case study. *Eos Transcripts of the American Geophysical Union*, 87(52): Fall Meeting Supplement, Abstract S51E–07, December 2006.

[42] T. Tu, L. Ramirez-Guzman, H. Yu, J. Bielak, O. Ghattas, K. Ma, and D. O'Hallaron. From mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing. In *Proc. Supercomputing (SC2006)*, Tampa, FL, November 2006. ACM/IEEE.

[43] Tiankai Tu, David O'Hallaron, and Julio López. Etree – a database-oriented method for generating large octree meshes. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 127–138, Ithaca, NY, Sep 2002.

[44] Tiankai Tu, David R. O'Hallaron, and Omar Ghattas. Scalable parallel octree meshing for terascale applications. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 4, Washington, DC, USA, 2005. IEEE Computer Society.

[45] Tiankai Tu, Hongfeng Yu, Jacobo Bielak, Omar Ghattas, Julio López, Kwan-Liu Ma, David R. O'Hallaron, Leonardo Ramírez-Guzmán, Nathan Stone, Ricardo Taborda, and John Urbanic. Remote runtime steering of integrated terascale simulation and visualization. In *SC '06: Proceedings of the 2006 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Tampa, FL, November 11–17, 2006.

[46] Tiankai Tu, Hongfeng Yu, Leonardo Ramírez-Guzmán, Jacobo Bielak, Omar Ghattas, Kwan-Liu Ma, and David R. O'Hallaron. From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *SC '06: Proceedings of the 2006 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, page 15, Tampa, Florida, November 2006. IEEE Computer Society.