Earthquake Ground Motion Modeling on Parallel Computers

Hesheng Bao
Computational Mechanics Laboratory, Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`hbao@cs.cmu.edu`
`http://www.cs.cmu.edu/afs/cs.cmu.edu/user/hbao/www/home.html`

Jacobo Bielak
Computational Mechanics Laboratory, Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`bielak@cs.cmu.edu`
`http://www.ce.cmu.edu/user/faculty/bielak.html`

Omar Ghattas
Computational Mechanics Laboratory, Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`oghattas@cs.cmu.edu`
`http://www.cs.cmu.edu/~oghattas`

Loukas F. Kallivokas
Computational Mechanics Laboratory, Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`loukas@cs.cmu.edu`

David R. O'Hallaron
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
`droh@cs.cmu.edu`
`http://www.cs.cmu.edu/~droh/`

Jonathan Richard Shewchuk
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
`jrs@cs.cmu.edu`

Jifeng Xu
Computational Mechanics Laboratory, Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`jxu@cs.cmu.edu`
`http://www.cs.cmu.edu/afs/cs/user/jxu/www/documents/home.html`

Abstract. We describe the design and discuss the performance of a parallel elastic wave propagation simulator that is being used to model earthquake-induced ground motion in large sedimentary basins. The components of the system include mesh generators, a mesh partitioner, a parceler, and a parallel code generator, as well as parallel numerical methods for applying seismic forces, incorporating absorbing boundaries, and solving the discretized wave propagation problem. We discuss performance on the Cray T3D for unstructured mesh wave propagation problems of up to 77 million tetrahedra. By paying careful attention to each step of the process, we obtain excellent performance despite the highly irregular structure of the problem. The mesh generator, partitioner, parceler, and code generator collectively form an integrated toolset called Archimedes, which automates the solution of unstructured mesh PDE problems on parallel computers, and is being used for other unstructured mesh applications beyond ground motion modeling.

Keywords. absorbing boundaries, computational geometry, finite element methods, local site effects, mesh generation, parallel unstructured PDE solvers, parallelizing compilers, seismic wave propagation, strong ground motion.

1 Introduction

The reduction of the earthquake hazard to the general population is a major problem facing the U.S. and other countries. To this end, it is essential that within earthquake-prone regions new facilities be designed to resist earthquakes and existing structures be retrofitted as necessary. Assessing the free-field ground motion to which a structure will be exposed during its lifetime is a critical first step in the design process. Ground motion is usually specified through seismic design spectra, which essentially prescribe an equivalent lateral force that the structure must withstand without failure. This force is based upon (i) past seismic history in the general geographic location, (ii) position with respect to possible earthquake sources such as active faults, (ii) expected earthquake magnitudes, and (iv) general geologic conditions.

Observations of ground motion during recent strong earthquakes have shown, however, that three-dimensional local site effects, which are normally given only passing attention in design, can be extremely significant, and can adversely affect structural safety. Three common effects often observed in basins or sedimentary valleys are an amplification and significantly longer duration of the surface ground motion with respect to that in rock. In addition, there is a more rapid spatial variation of the ground motion that can cause large differential base motion of extended structures such as bridges or dams.

Examples of these effects are plentiful. Perhaps the most dramatic recent occurrences are those in Mexico City in 1985 and within the San Francisco Bay area during the 1989 Loma Prieta earthquake. For both of these events and their aftershocks, amplifications greater than 4 or 5 and durations of up to 15 to 30 seconds greater than the corresponding motion on rock were quite common, due to local site conditions. Studies of these and other earthquakes indicate that the presence of large sediment-filled basins significantly amplifies the strength of the waves observed within the basins.

It is now generally recognized that while one- and two-dimensional local models can help explain observed behavior in certain situations, a complete quantitative understanding of strong ground motion in large basins requires a simultaneous consideration of three-dimensional effects of earthquake source, propagation path, and local site conditions. See [1] for a general overview, and [12, 11, 22, 8, 9, 18, 16], for instance, for representative recent work in this field. The large scale associated with modeling strong ground motion in large basins places enormous demands on computational resources, and renders this problem among the “Grand Challenges” in high performance computing.

2 Numerical approximation issues

Simulating the earthquake response of a large basin is accomplished by numerically solving the partial differential equations (PDEs) of elastic wave propagation, i.e. Navier’s equations of elastodynamics. A variety of numerical methods have been used for approximating the solution of these problems. While boundary element methods have been popular for moderately-sized linear models, the inhomogeneity, nonlinearity, and large scale of such basins as the Greater Los Angeles Basin preclude their use here. On the other hand, uniform grid domain methods such as finite differences become impractical for the very large problem sizes involved.

To see why uniform grids are impractical, consider the Los Angeles Basin. For a shear wave velocity of 0.4 km/s and a frequency of 2 Hz, a regular discretization of the elasticity operator would place grid points 0.02 km apart to achieve second order accuracy. The region of interest has dimensions 140 km$\times$100 km$\times$20 km; thus, a regular discretization, governed by the softest layer, requires 35 billion grid points with three displacement components per grid point. At least a terabyte of primary memory would be needed, and on the order of 10^{13} operations would be required at each time step. The stability condition associated with explicit time integration of the semidiscrete equations of motion imposes a time increment at least as small as 0.004s. Thus, a computer would have to perform at a sustained teraflop per second for two days to simulate a minute of shaking.

Instead, we use unstructured mesh finite element methods that tailor the mesh size to the local wavelength of propagating waves. The wavelength is given by the product of the shear wave velocity and the period of excitation. The shear wave velocity is a property of the soil type; for a basin such as Los Angeles, it varies from 0.22 km/s to 4.5 km/s throughout the basin. Since in three dimensions mesh density varies with the cube of shear wave velocity, and since the softest soils are concentrated in the top layers, this means that an unstructured mesh method may yield three orders of magnitude fewer equations than with structured grids. Modeling the Los Angeles Basin for values of earthquake period and wave velocity that are desirable for engineering purposes thus becomes practical on the largest of today's parallel supercomputers.

We favor finite element methods for their ability to efficiently resolve multiscale phenomena, the ease with which they handle stress boundary conditions, and their firm theoretical foundation. For temporal approximation, we have studied both explicit and preconditioned conjugate gradient-based implicit methods. For hyperbolic problems, explicit methods become unstable if the time step is greater than the time it takes an elastic wave to cross any element—the Courant condition. Implicit methods, on the other hand, are unconditionally stable, implying that larger time steps can be taken. However, the very characteristic that makes them stable—the fact that the solution at time $t + \Delta t$ requires information from all nodes[1] at time t—renders them unattractive on distributed memory computers, since this implies global information exchange. We have found that our mesh generators give us such good control over mesh resolution that the Courant condition for explicit methods is not onerous. The result is that the more readily parallelizable explicit methods perform better for elastic wave propagation problems. In this paper we discuss only a single-step explicit time integration method.

While unstructured mesh methods for simulating wave propagation through heterogeneous media result in many fewer equations, they introduce a number of computational difficulties that must be overcome. First, mesh resolution must closely follow wavelength; too coarse a resolution will introduce error, too fine will result in unnecessary computation as well as excessively small time steps (when explicit integration methods are used). Second, element aspect ratios must remain small; large aspect ratios will eventually result in instability in the time integration scheme. Highly heterogeneous basins, in which wavelengths vary rapidly in space, introduce special difficulties when trying to follow the wavelength change without severely stretching the mesh. Third, unstructured mesh methods are not easy to program on parallel computers; their irregular data structures require nontrivial mappings onto parallel machines, and irregular communication patterns are generated. Thus, we have had to develop fast, robust computational geometry and mesh generation techniques for highly spatially-variable meshes as well as compilers and tools that simplify the programming of unstructured mesh methods on parallel systems.

[1] In this paper, we use the term *node* in the finite element context, i.e. a vertex in the finite element mesh.

In the remainder of this paper, we present numerical methods, algorithms, and implementations for modeling earthquake-induced ground motion modeling in highly heterogeneous basins, and give performance results on the Cray T3D. We also describe Archimedes, a toolset/compiler we have built for automated solution of unstructured mesh PDE problems on parallel distributed memory computers. For an alternative approach to parallel ground motion modeling on distributed memory machines, see the work of Olsen, Archuleta, and Matarese [17], which employs finite differences on regular grids. See also the references to prior finite difference modeling work on sequential machines contained therein. In addition to the finite element method we describe in this paper, there have been recent efforts to endow finite difference wave propagation methods with multi-resolution capabilities. See the work described in [14], which uses composites of regular grids to achieve variable resolution.

3 Algorithms

In this section we discuss the numerical and geometric algorithms necessary for modeling earthquake-induced ground motion in large, heterogeneous basins. In the next three subsections, we briefly discuss a mesh generation technique capable of resolving local wavelengths, a mesh partitioner that rapidly provides asymptotically optimal partitions, and several initialization steps that are carried out prior to parallel solution of the discrete wave propagation equations. The last subsection provides the governing equations and overviews spatial and temporal discretization strategies appropriate for distributed memory parallel computers.

3.1 Mesh generation

As we have seen, seismic wave propagation problems place special demands on mesh generators, including the need for tight control over mesh resolution and aspect ratio, and the need to support extremely large problem sizes. We have developed a fast, stable, and efficient meshing algorithm for generating very large scale meshes, suitable for the large basins we target. Since repeated computations will be performed with a single mesh (one or two dozen earthquake scenarios, each involving thousands of time steps), we have decided to generate and partition each mesh sequentially. However, care must be taken in designing and implementing efficient algorithms for these steps, lest they become bottlenecks.

Mesh generation begins with a database of the material properties of the soil and rock within and around a particular basin. These material properties—the shear wave velocity, the dilatational wave velocity, and the density—are estimated throughout the basin from soil borings, from geological records, and from seismic prospecting studies. Figure 1 shows the variation in shear wave velocity at a depth of three meters from the valley fill surface in a region surrounding the San Fernando Valley in Southern California. The material property model on which this image is based was provided by H. Magistrale and S. Day at San Diego State University [13]. The figure shows a variation in shear wave velocity of at least a factor of seven. Since element width is inversely proportional to velocity, a regular grid method can have up to $7^3 = 343$ times more points per unit volume than an unstructured mesh for this material model.

The meshing algorithm comprises two steps. First, we generate an octree that resolves the local wavelength of shear waves. The wavelength is known from the shear wave velocity and the frequency of excitation. Based on numerical experiments with homogeneous problems and on some theoretical analysis, we have found that 8–10 nodes per wavelength is sufficient for "engineering", or 95%, accuracy when using linear finite elements. (In Section 3.4 we make precise what we mean by 95% accuracy.) When constructing the octree, we enforce the rule that adjacent cells may not differ in edge length by more than a factor of two, producing a *balanced* octree. This is crucial for producing elements with bounded aspect ratios. Bounding

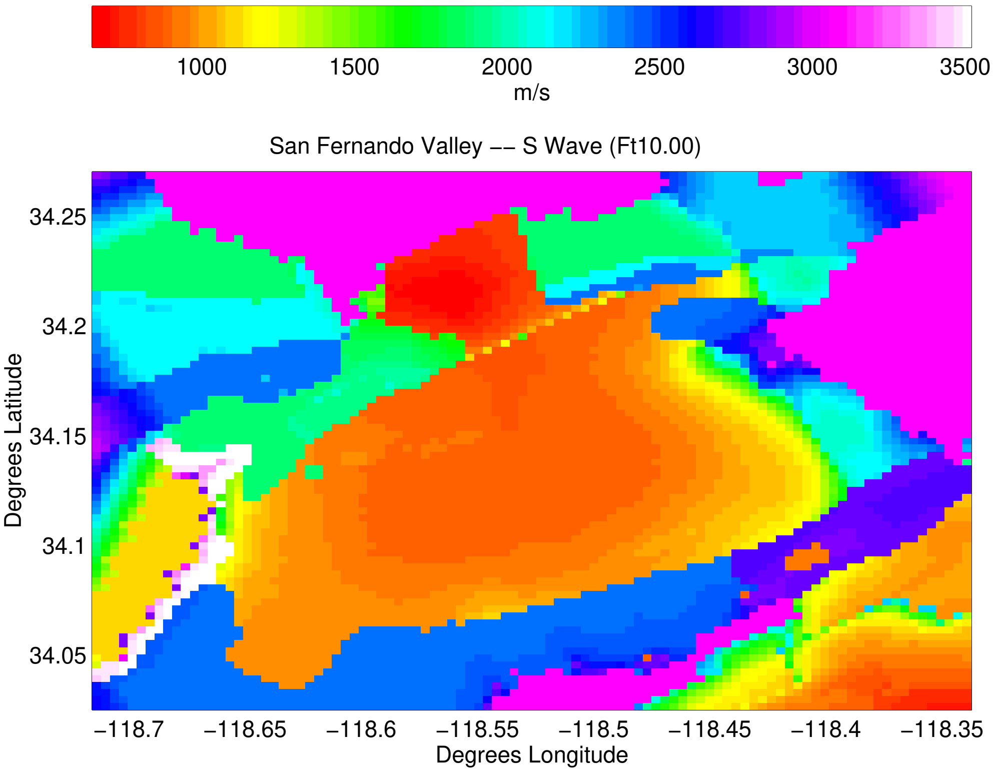

Figure 1: Surface distribution of shear wave velocity in the San Fernando Valley.

the aspect ratio of elements is important because aspects ratios far from one lead to poorly conditioned stiffness matrices, which can lead to instability in time integration.

Once a balanced octree is created such that no cell is wider than one-tenth the length of the wave that passes through it, a finite element node is placed at each cell vertex. Figure 2 depicts the nodes generated by the balanced octree for the San Fernando Basin properties of Figure 1. This set of nodes is then triangulated (more properly, tetrahedralized) according to the Delaunay criterion.[2] Figure 3 shows the resulting mesh of tetrahedra. Delaunay tetrahedralization is performed by a straightforward implementation of the Bowyer/Watson incremental algorithm [5, 23], which constructs a triangulation by adding one node at a time and locally adjusting the mesh to maintain the Delaunay criterion.

We have found that the Bowyer/Watson algorithm is occasionally sensitive to floating-point roundoff error; tetrahedral mesh generation can fail dramatically because of roundoff when processing near-degenerate geometric features. Such failures became increasingly common for us as the size of our meshes grew. To overcome this problem, we have developed a method for fast exact arithmetic that is particularly well-suited for certain tests that arise in computational geometry codes [19]. Our method is used to construct predicates that determine whether a point falls to the left or right side of a line, or whether a point falls inside or outside a sphere. These predicates are adaptive in the sense that they only use exact arithmetic to the extent it is needed to ensure a correct answer. Hence, if a point falls very close to a line, high precision arithmetic may be needed to resolve which side of the line it falls on; if a point is far from a line, approximate arithmetic will suffice, so the test can be performed quickly. Because the latter case is far more common, our exact arithmetic predicates are on average only slightly slower than ordinary, nonrobust floating-point predicates,

[2]We could have used a hexahedral mesh directly from the octree, but the elements would have required special treatment to make them conforming.

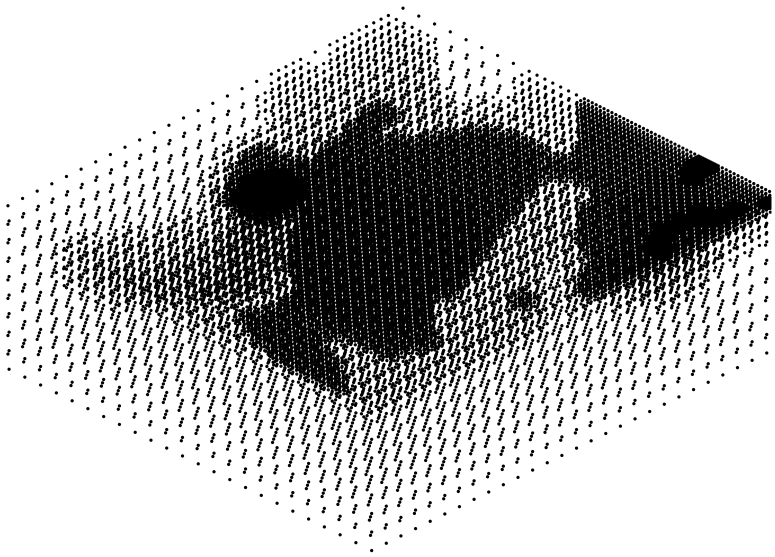

Figure 2: Nodal distribution for the San Fernando Valley. Node generation is based on an octree method that locally resolves the elastic wavelength. The node distribution here is a factor of 12 coarser in each direction than the real one used for simulation, which is too fine to be shown, and appears solid black when displayed. However, the relative resolution between soft soil regions and rock illustrated here is similar to that of the 13 million node model we use for simulations.

and our Delaunay tetrahedralization code runs quickly while ensuring the integrity of its results.

Our use of the Delaunay tetrahedralization of the vertices of a balanced octree guarantees that element aspect ratios are bounded, and that element sizes are chosen appropriately so that wavelengths are sufficiently resolved without unnecessary resolution (provided the material properties do not vary too rapidly).

We have used our mesh generator to create a mesh of the San Fernando Basin with a 220 m/s shear wave velocity in the softest soil for an earthquake with a highest frequency of 2 Hz. The mesh contains 77 million tetrahedra and 13 million nodes, and was generated in 13 hours on one processor of a DEC 8400, requiring 7.7 Gb of memory. It has a maximum aspect ratio of 5.5 and exhibits a spatial resolution variability of an order of magnitude. This mesh is perhaps the largest unstructured mesh generated to date.

3.2 Mesh partitioning

Once a mesh is generated, the set of elements that comprise it must be partitioned into subdomains. Each subdomain can then be mapped onto a processor of a parallel machine. The goal of mesh partitioning is to minimize communication time while maintaining load balance. In an explicit method, communication is associated with the nodes that lie on the boundaries between subdomains and are shared by more than one processor. Processors sharing a node must communicate six words per shared node for each matrix-vector multiply, i.e. twice each time step in our method. Communication time depends on both the message sizes, which increase with the number of shared nodes, and the number of messages, which increases with the number of adjacent subdomains. The load on a processor for explicit solution of linear wave propagation

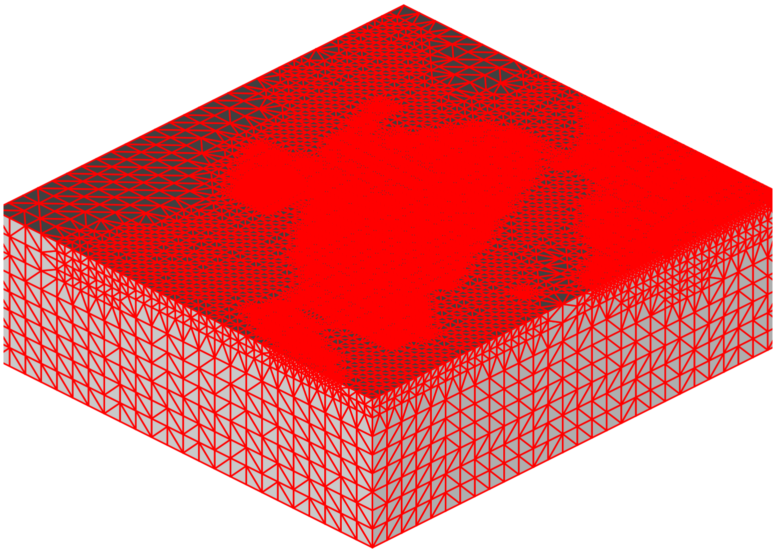

Figure 3: Tetrahedral element mesh of the San Fernando Valley. Maximum tetrahedral aspect ratio is 5.5. Again, the mesh is much coarser than those used for simulation, for illustration purposes.

problems is easy to predict: it is proportional to the number of nodes on that processor. Prediction becomes more difficult when nonlinearities are present, such as with the soil plasticity models that we are currently introducing into our code. In these cases, the work per node is solution-dependent. Nevertheless, for our purposes, we consider a mesh partitioner desirable if it produces subdomains of nearly equal size (where size is measured by number of elements and not by volume) and with as few nodes shared between processors as is reasonably possible.

The partitioner we use is based on the algorithm of Miller, Teng, Thurston, and Vavasis [15]. This algorithm uses geometric information to construct a separator, i.e. a set of nodes whose removal separates the mesh into two pieces of roughly equal size. Each of these pieces is then recursively partitioned until the desired number of subdomains is reached. The Miller et al. algorithm produces separators that are asymptotically optimal; their length is of order $O(N^{2/3})$ in three dimensions, where N is the number of nodes. Theoretically, the algorithm runs in randomized linear time; in practice, the algorithm rapidly produces high quality partitions.

As an illustration, our implementation of this algorithm partitioned the 77 million element mesh described above into 256 subdomains in about 3.8 hours on one processor of the DEC 8400, and required 7.9 Gb of memory. The resulting partition (again for a factor of twelve coarser mesh) is shown in Figure 4. The figure shows the circular cuts produced by the partitioner. Despite the high spatial variability of the mesh, the partitions appear to be well-shaped.

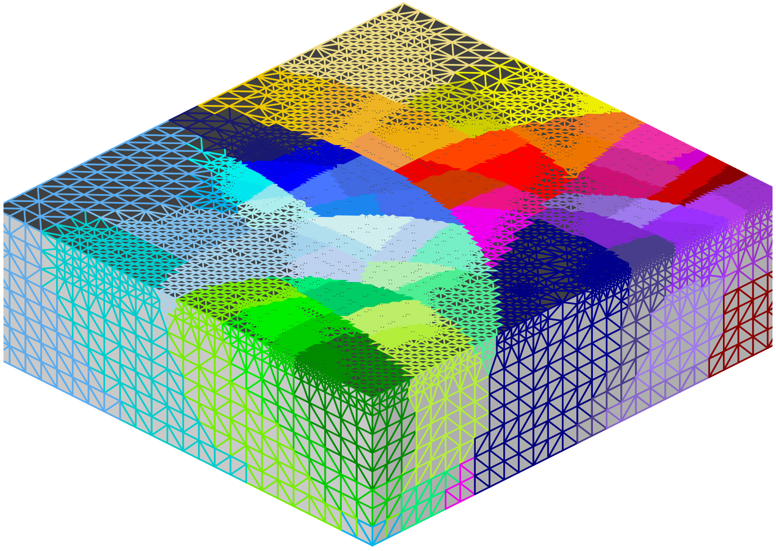

Figure 4: Mesh partitioned for 64 subdomains.

3.3 Parceling

After a mesh is partitioned into subdomains, there remain several operations that have to be performed on the partitions to prepare the input for the parallel program. We refer to these steps collectively as *parceling*. The steps include generating (i) the communication schedule for each processor, (ii) the global-to-local mapping information, which allows identification of a node or element number on a processor by its global number, and (iii) the nonzero structure of the stiffness matrix on each processor. The last item could be performed in parallel, but it takes little time and provides us with useful statistics on the mesh, so we perform it sequentially. Parceling requires about 2.3 hours and 7.7 Gb memory on the DEC 8400 for the 77 million element San Fernando Basin mesh. The communication graph generated by the parceler is shown in Figure 5. Each vertex represents a subdomain and corresponding processor; each edge represents communication between two processors.

3.4 Governing equations and discretization

Whereas the mesh generation, partitioning, and parceling steps are currently performed sequentially, the wave propagation equations are solved on a parallel machine. This section describes the numerical techniques we use. Navier's equations of elastodynamics for an isotropic, heterogeneous medium are

$$\nabla \cdot \left[\mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right) + \lambda \left(\nabla \cdot \mathbf{u}\right) \mathbf{I}\right] = \rho \frac{\partial^2 \mathbf{u}}{\partial t^2}, \tag{1}$$

where $\mathbf{u}$ is the displacement vector field, ρ is the density, and λ and μ are elastic material constants, which depend on the shear and dilatational wave velocities.

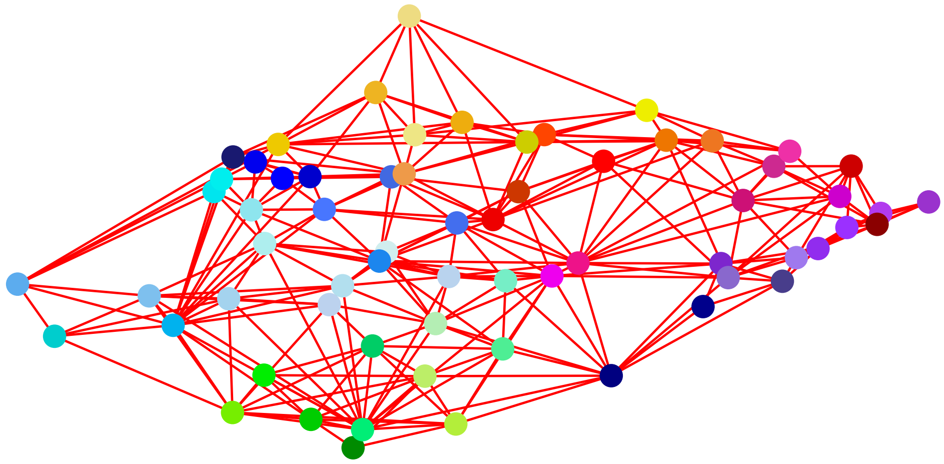

Figure 5: Communication graph for the partitioned element mesh depicted in Figure 4.

The domain of the problem is an elastic halfspace, i.e. semi-infinite. In order to render the computational domain finite, we introduce absorbing boundaries at the bottom and sides of the basin that are local in both space and time [4, 10]. These boundaries allow the passage of outgoing waves with minimum reflection.

Since, in many cases, the earthquake source can be outside the computational domain, its effect must be introduced into the region. This is carried out as described in [3, 6] by means of effective forces. In short, for an arbitrary earthquake excitation these forces are determined in terms of the free-field motion by introducing a fictitious auxiliary surface that surrounds the basin. Across this auxiliary surface one imposes the conditions of continuity of displacement and traction. By selecting the total displacement vector field as the unknown in the resulting interior region and the scattered displacement field in the exterior region, the free-field displacement and traction now appear explicitly in the continuity conditions, which become jump conditions, with the free-field displacement and traction on the righthand side. These non-homogeneous terms on the righthand side are the ones that give rise to the effective forces upon spatial discretization. If, on the other hand, the seismic source is located inside the computational domain, say as a kinematic dislocation across the fault, one can select the fault itself as the auxiliary surface. The procedure is similar, but now one uses the total displacement everywhere as the unknown field; thus, the displacement field again experiences a jump across the interface, but the traction remains continuous. Notice that with this technique, whether the source is originally located inside or outside the computational domain, only outgoing waves will impinge upon the absorbing boundary. Both types of source are implemented in our code.

We also model material damping in the basin via viscous damping. With these modifications, standard Galerkin discretization in space by finite elements produces a system of ordinary differential equations of the form

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\mathbf{u} + \mathbf{K}\mathbf{u} = \mathbf{f}, \tag{2}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix associated with the absorbing boundary and material damping, $\mathbf{K}$ is the stiffness matrix, and $\mathbf{f}$ is the effective force vector. Here, $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are block matrices; the (i, j)th block of $\mathbf{M}$ is a 3×3 matrix given by

$$\mathbf{M}_{ij} = \rho \int_{\Omega} \phi_i \phi_j \mathbf{I} \, d\Omega, \tag{3}$$

and the (i, j)th block of $\mathbf{K}$ is given by

$$\mathbf{K}_{ij} = (\mu + \lambda) \int_{\Omega} \nabla\phi_i \nabla\phi_j^T \, d\mathbf{\Omega} + \mu \int_{\Omega} \nabla\phi_i^T \nabla\phi_j \mathbf{I} \, d\mathbf{\Omega}, \tag{4}$$

where ϕ_i is the finite element global basis function associated with the i-th node.

Damping is introduced through a proportional damping approximation at the element level, i.e. we take

$$\mathbf{C}^e = \alpha\mathbf{M}^e + \beta\mathbf{K}^e, \tag{5}$$

where α and β are scalar constants and the superscript e indicates an element matrix. The first term leads to as damping factor that is inversely proportional to frequency, and the second to one that is linear in frequency. The constants α and β, which may vary within the basin according to the type of material, are chosen to best fit a prescribed attenuation law.

Given appropriate initial conditions, this system of ODEs can be integrated in time using central differences, yielding the explicit method

$$\left(\mathbf{M} + \frac{\Delta t}{2}\mathbf{C}\right) \mathbf{u}_{t+\Delta t} = \Delta t^2 \mathbf{f}_t - \left(\Delta t^2 \mathbf{K} - 2\mathbf{M}\right) \mathbf{u}_t - \left(\mathbf{M} - \frac{\Delta t}{2}\mathbf{C}\right) \mathbf{u}_{t-\Delta t}. \tag{6}$$

This method exhibits second-order accuracy in time, and when coupled with linear finite elements, we obtain second-order accuracy in space as well. We use a lumped mass approximation to $\mathbf{M}$, which amounts to numerically integrating (3) with integration points at element vertices. This results in a diagonal mass matrix. To render the lefthand side operator of (6) diagonal, we further evaluate the off-diagonal components of $\mathbf{C}$ at $\mathbf{u}_t$ rather than $\mathbf{u}_{t+\Delta t}$. Inversion of the time stepping operator thus requires only a scaling of the righthand side of (6), which is carried out just once prior to time stepping. Forming the products of $\mathbf{K}$ and $\mathbf{C}$ with vectors comprises the major computational effort associated with iterating on (6). The sparsity structure of $\mathbf{K}$ is dictated by the underlying finite element mesh, and is thus very irregular. If shear waves are not over-resolved, the time step necessitated by stability is of the order of the time step dictated by accuracy, which is what an implicit method would take. By choosing an explicit method we avoid solving linear systems at each time step. Thus, overall, the explicit method is superior for our application, especially on a parallel computer.

We have tried several different choices of basis function order, and have concluded that piecewise-linear functions are the most efficient for problems requiring engineering accuracy. Our conclusion is based on numerical experimentation using plane Ricker wavelets on unstructured homogeneous meshes (in which case we know what the exact solution should be), but a simple argument can be given as follows. We recognize first that (spatial) approximation errors are bounded from above by interpolation errors. We then ask, for a given order of basis function and a given acceptable level of infinity-norm error, how many nodal points are required to produce a piecewise-polynomial interpolant of a simple harmonic wave. Next, we convert the required number of nodes per wavelength to an estimate of the storage and work required for an iteration of the explicit method (6). For example, if N is the total number of nodes, one can show that trilinear hexahedra require $163.5N$ words of storage and $498N$ flops/time step, while triquadratic hexahedra necessitate $367N$ words and $1164N$ flops/time step. So, for example, triquadratic elements should require at least 2.2 times fewer nodes in order to be preferred (for storage reasons) over trilinear elements. However, one-dimensional interpolation theory tells us that 5% error requires 10 nodes per wavelength using linear elements or 9.4 nodes using quadratics. Thus, in three dimensions, triquadratics only allow $(10/9.4)^3 = 1.2$ times fewer nodes than trilinears, and are thus not warranted. An opposite conclusion is reached if one demands 99% accuracy. Our confidence in the values of material properties and in the fidelity of the source models for this problem does not warrant solution accuracies greater than 95%. Thus, we conclude that for this level of accuracy, the powers of higher-order interpolation are offset by their increased cost, both in storage and in increased work.

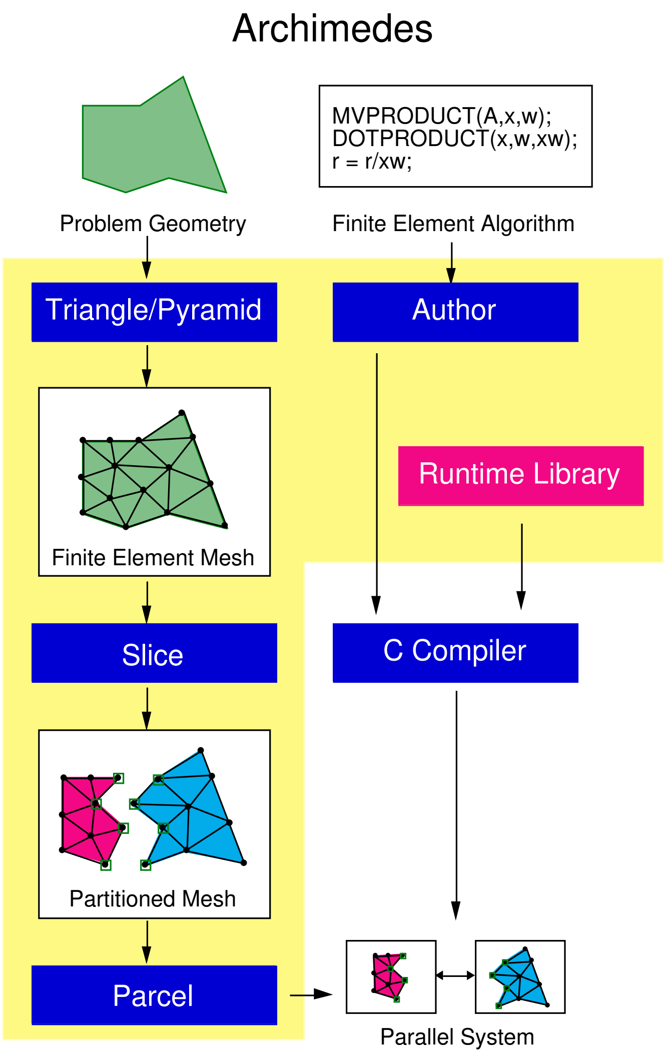

Figure 6: The Archimedes system.

4 Archimedes: A system for unstructured PDE problems on parallel computers

Archimedes [7, 2] is a general-purpose toolset for the efficient mapping of unstructured mesh computations arising from numerical solution of PDEs onto parallel systems. Archimedes is designed to insulate the user from issues of parallelism and communication, and to allow easy and natural expression of unstructured mesh finite element methods and solvers. Its component tools are based on the algorithms for mesh generation, partitioning, and parceling described above. Archimedes also includes a code generator called Author that is targeted to the sorts of computations arising in the solution of PDE problems by unstructured finite element methods [21].

The Archimedes system is depicted in Figure 6. Input to Archimedes includes (i) the problem geometry, and (ii) a sequential program containing an element-level description of the finite element approximation as well as a high-level description of the solution method. The input program is written in C, augmented with finite element-specific and linear algebraic primitive operations that include vector and matrix assembly, imposition of boundary conditions, sparse matrix-vector products, dot products, and preconditioning. Additional functions are specific to elastic wave propagation, and include absorbing boundaries, damping, and seismic input incorporation. Archimedes programs contain no explicit communication statements, and thus can be written without any knowledge of the parallel machine's underlying communication system. The set of primitives that Archimedes understands is rich enough to express algorithms for solution of linear and nonlinear scalar and vector PDEs, using arbitrary-order finite elements in space and either explicit or implicit methods in time. For implicit methods, the Archimedes language provides for expression of various stationary iterative solvers as well as Krylov subspace methods. Furthermore, users can add new primitives as the need arises.

Triangle [20] is a two-dimensional quality mesh generator in the Archimedes toolset. Triangle operates on a description of the input geometry and produces a two-dimensional triangular mesh with guaranteed angle bounds that satisfies user-specified bounds on element size. These element size bounds can vary spatially, and can be set *a priori*, based on features of the problem geometry, or *a posteriori*, within a solution-adaptive scheme. Archimedes also includes a rudimentary three-dimensional mesher, Pyramid. We use Pyramid's Delaunay capability to tetrahedralize the node sets generated by the octree algorithm, as described in Section 3.1. This is sufficient for basin meshes, since the geometry is simple but the physics drives the mesh resolution. Other finite element applications, for example solid mechanics and aerodynamics, will require support for more complex geometries. Extensions to Pyramid to allow for arbitrary geometry are underway. Sequential mesh generation as we have implemented it in Archimedes on the DEC 8400 is adequate for up to 100–200 million tetrahedra. For the largest problems, such as the Greater Los Angeles Basin at a frequency of 2Hz, sequential mesh generation may become a bottleneck. We may parallelize this task in the future.

Archimedes' toolset also includes Slice, an implementation of the asymptotically optimal mesh partitioner [15] discussed in Section 3.2. Once Slice partitions a mesh into subdomains, its output is fed to Parcel, which prepares input for the parallel program. Parcel generates the communication schedule for each processor, the global-to-local mapping information for the mesh nodes and elements, and the local stiffness matrix structure. As with mesh generation, sequential execution (on the DEC 8400) of the mesh partitioning and parceling tasks is sufficiently quick for the large scale problems we target. For problems requiring more than 100–200 million elements, we may have to parallelize partitioning and parceling, more for memory reasons than speed. Unlike meshing, parallelizing these two tasks is relatively straightforward.

The output of Parcel is a single file containing mesh information; this is read by the parallel program. The only program output we are interested in regards free surface displacements and velocities, rather than volume information; thus output is not as significant a problem here as it is in such applications as fluid mechanics. In the interest of portability, we have not yet parallelized I/O. As problem size continues to grow in the future, parallel I/O may become necessary.

Archimedes' parallelizing compiler generates code for any parallel system with C and MPI implementations, including networks of workstations (using the Argonne/Mississippi State MPICH implementation), Intel's Paragon (also using MPICH) and the Cray T3D (using the CRI/EPCC MPI implementation). Finally, Archimedes includes Show Me, an X-Windows based unstructured mesh visualization system. This allows 3D display of nodal distributions, meshes, partitions, communication graphs, and solutions. Such basic capabilities as rotation, shading, zooming, cutaway views, and PostScript output are supported. Figures 2, 3, 4, and 5 were generated by Show Me.

Our decision to build Archimedes was undertaken for several reasons. First, such a system allows application specialists to focus on what they do best: designing numerical methods and concentrating on the physics of the problems at hand. Archimedes also ensures that their simulations will still be running when today's parallel hardware is obsolete. Indeed, earthquake engineering students in our lab have been writing Archimedes parallel programs and running them on the T3D without any concern for the underlying parallel hardware or system software. This insulation has not come at the price of performance; we regularly observe 30 megaflops per processor on the T3D, which is quite good for irregular sparse matrix calculations.

The second reason for creating Archimedes is that it eases the process of prototyping different numerical algorithms. With the basic library of primitives in place, we can quickly experiment with different time integration schemes, preconditioners, and element types. During the course of our research, we studied implicit versus explicit methods, lumped versus consistent mass matrices, first-order versus second-order absorbing boundaries, linear versus quadratic finite elements, and bubble-mode-enhanced versus standard Lagrange elements. The ability to express numerical algorithms in an intuitive, sequential manner was crucial in allowing us to study the implications of our numerical decisions, before we settled on our current choices. The functionality of the Archimedes language continues to grow in response to new algorithmic needs.

Our final motivation in designing Archimedes is that a number of the numerical and computational issues faced in modeling earthquake-induced ground motion are shared by many other applications in computational science and engineering. Unstructured mesh methods are useful for PDE problems that are characterized by complex geometries or that exhibit multiscale phenomena, such as transonic flows, crack propagation, large deformation materials processing, and pollutant transport, or . Our goal was to make Archimedes useful for this broader class of problems. Indeed, Archimedes is now being used in several areas other than ground motion modeling.

Many researchers do not wish to bother with low level details of programming a parallel machine, yet still want the efficiency associated with message-passing code. The Archimedes code generator is designed so that it can be extended by users without having to rebuild the system. For example, users can write their own parallel preconditioner routines and register them with the Archimedes compiler without recompiling any code. This provides a mechanism for the system to grow and evolve.

5 Performance on the Cray T3D

In this section we provide timings that characterize the performance of our parallel explicit wave propagation code on the Cray T3D. We are currently using the code to study the earthquake-induced dynamics of the San Fernando Valley in Southern California. The San Fernando simulations involve meshes of up to 77 million tetrahedra and 40 million equations. The largest mesh corresponds to the case of a lowest shear wave velocity of 220 m/s and a maximum frequency of 2 Hz; the code requires nearly 16 Gb of memory and takes 6 hours to execute for 40,000 times steps on 256 processors of the Cray T3D at the Pittsburgh Supercomputing Center (PSC). Results of a typical simulation, in which the basin was subjected to a vertically-incident plane wave Ricker pulse with a central frequency of 1 Hz, indicate a factor of amplification of eight in the softer parts of the basin compared to the maximum displacement on rock. This suggests substantially greater damage in these regions. A typical result is shown in Figure 7, which depicts the amplification induced by the soft soil. Simulations of this type are essential to better predict the local site effects within soft basins such as those on which Los Angeles, San Francisco, Mexico City, and Tokyo are situated.

The relevant scenario for assessing the performance of our earthquake simulations as the number of processors increases is one in which the problem size increases proportionally, because unstructured PDE problems are typically memory-bound rather than compute-bound. Given a certain number of processors, we typically aim at full use of their memory; as the number of processors increases, we take advantage of their additional memory by increasing the problem size. In order to study the performance of our earthquake ground motion simulation code with increasing problem size, we have generated a sequence of increasingly-finer meshes for the San Fernando Basin. These meshes are labeled `sf10`, `sf5`, `sf2`, and `sf1`, and correspond to earthquake excitation periods of 10, 5, 2, and 1 second, respectively. Additionally, the mesh `sf1b` corresponds to a geological model that includes much softer soil in the top 30m, and thus

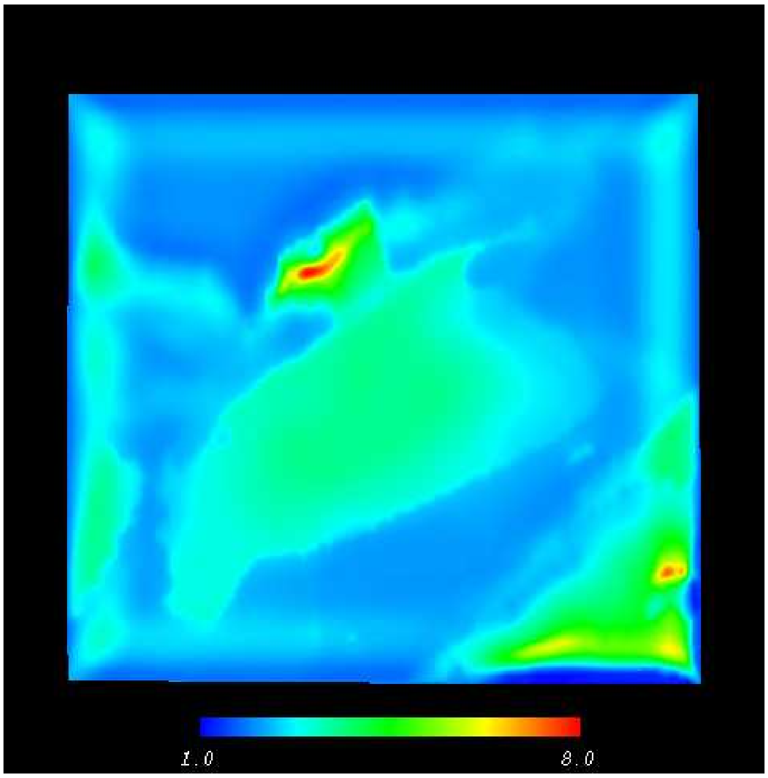

Figure 7: Surface distribution of ground motion amplification factors in the San Fernando Valley. The amplification factors have been calculated by comparing the surface to the bedrock motion.

necessitates an even finer mesh. Note that mesh resolution varies with the inverse cube of period, so that halving the period results in a factor of eight increase in the number of nodes. Characteristics of the five meshes are given in Table 1.

Our timings include computation and communication but exclude I/O. We exclude I/O time because in our current implementation it is serial and unoptimized, and because the T3D has a slow I/O system. I/O time involves the time at the beginning of the program to input the file produced by Parcel, as well as the time to output results every tenth time step to disk. With the availability of the Cray T3E at PSC, we plan to address parallel I/O in the future.

We begin with a traditional speedup histogram, for which the problem size is fixed and the number of processors is increased. Figure 8 shows the total time, as well as the relative time spent for communication and computation, for an earthquake ground motion simulation, as a function of the number of processors.

Table 1: Characteristics of San Fernando Basin meshes.

mesh	nodes	equations	elements
`sf10`	7,924	21,882	35,047
`sf5`	30,169	90,507	151,173
`sf2`	378,747	1,136,241	2,067,739
`sf1`	2,461,694	7,385,082	13,980,162
`sf1b`	13,422,563	40,267,689	76,778,630

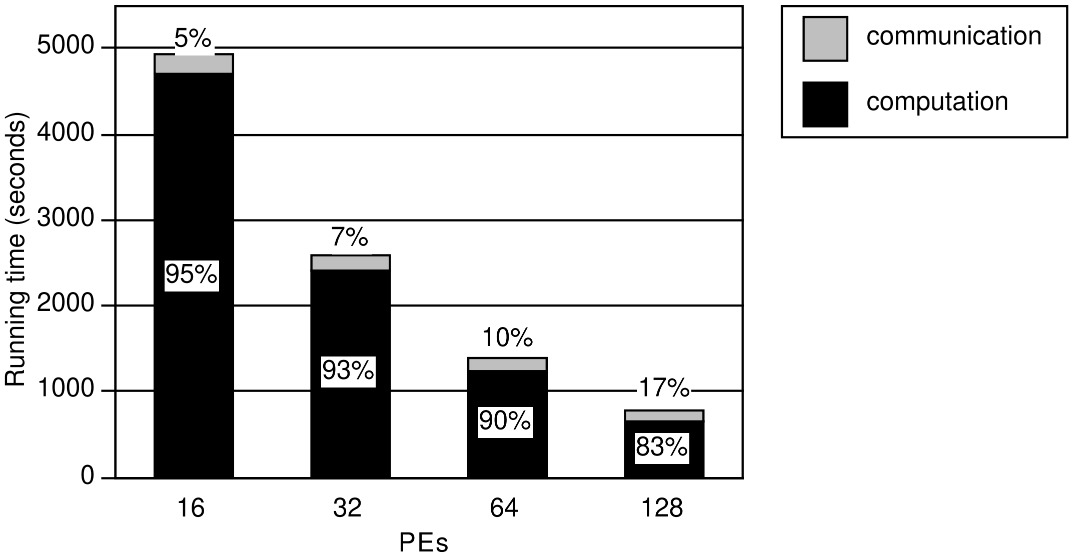

Figure 8: Timings in seconds on a Cray T3D as a function of number of processors (PEs), excluding I/O. The breakdown of computation and communication is shown. The mesh is `sf2`, and 6000 time steps are carried out.

The mesh used for these timings is `sf2`. On 16 processors, the time spent for communication is 5% of the time spent for computation, which is quite good for such a highly irregular problem. There are about 24,000 nodes per processor, which results in about half the memory on each processor being used. As the number of processors doubles, the percentage of time spent communicating relative to computing increases, as expected. For 128 processors, the communication time has increased to one-fifth of the total time. However, we are only utilizing $1/16$ of the local memory on a processor; practical simulations will generally exhibit performance more like the left bar of Figure 8.

We can quantify the decrease in computation to communication ratio for a regular mesh. Suppose there are N/P nodes on a processor, where P is the number of processors. Suppose further that the regular grid is partitioned into cubic subdomains of equal size, one to a processor. Since computation for an explicit method such as Equation 6 is proportional to the volume of nodes in a cube (subdomain), and communication is proportional to the number of nodes on the surface of the cube, the computation to communication ratio is proportional to $(N/P)^{1/3}$, i.e. the ratio of total nodes to surface nodes of the cube. Thus, for fixed N, the ratio is inversely proportional to $P^{1/3}$, at least for cubically-partitioned regular grids with large enough numbers of nodes per processor. Clearly, it is in our interest to keep N/P as large as possible, if we want to minimize communication time.

Let us extend this analysis to explicit methods on unstructured meshes. Suppose N/P remains constant for increasing N and P, i.e. the number of nodes per processor remains constant. Now suppose that we have a partitioner that guarantees that the number of interface nodes remains roughly constant as N and P increase proportionally. Then we can expect that the computation to communication ratio will remain constant as the problem size increases.[3] In this case, we have a method that scales linearly: the amount of time required to solve a problem that is doubled in size is unchanged if we double the number of processors. How close do we come to this ideal situation? First, we plot the log of the computation to communication ratio against the log of the number of processors, using the data in Figure 8. A least-squares fit yields a line with slope -0.39. For a regular grid with perfect partitioners, we have seen in the previous paragraph that

[3]To the extent that communication time is governed by the number of words communicated (as opposed to the number of messages, or to the route between communicating processors.)

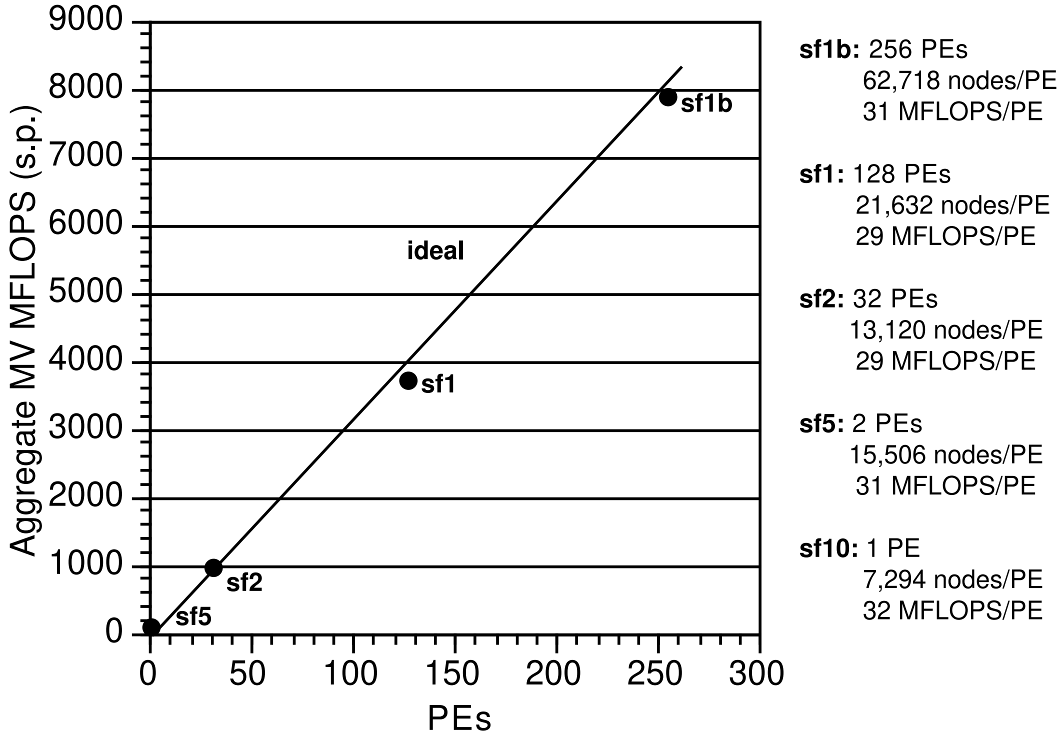

Figure 9: Aggregate performance on Cray T3D as a function of number of processors (PEs). Rate measured for matrix-vector (MV) product operations (which account for 80% of the total running time and all of the communication) during 6000 times steps.

this slope should be $-1/3$. This suggests that the idealized analysis is roughly applicable here.

Next, let us attempt to hold the number of nodes per processor roughly constant, and examine the aggregate performance of the machine as the problem size increases. It is difficult to maintain a constant value of N/P, since processors are available in powers of two on the T3D. However, we can still draw conclusions about scalability. Figure 9 shows the aggregate performance of our code on the T3D in megaflops per second, as a function of number of processors (and, implicitly, problem size). Megaflops are those that are sustained by matrix-vector product operations (which account for 80% of the total running time and all of the communication) during a San Fernando simulation, exclusive of I/O. This figure shows nearly ideal scalability, which is defined as the single processor performance multiplied by the number of processors. These results show that excellent performance is achievable, despite the highly heterogeneous mesh. This behavior requires a partitioner that keeps the number of interface nodes relatively constant for problem size that increases concomitantly with number of processors.

An even better measure of scalability is to chart the time taken per time step per node. If the algorithm/implementation/hardware combination is scalable, we expect that the time taken will not change with increasing problem size. Not only must the partitioner produce "scalable" partitions for this to happen, but in addition the PDE solver must scale linearly with N. This happens when the work per time step is $O(N)$. This is obvious from the iteration of Equation 6—vector sums, diagonal matrix inversions, and sparse matrix-vector multiplies require $O(N)$ operations.

Figure 10 depicts the trend in unit wall clock time as the number of processors is increased. Unit wall clock time is measured as microseconds per time step per average number of node per processor, which includes all computations and communications for all time steps, but excludes disk I/O. As we have said

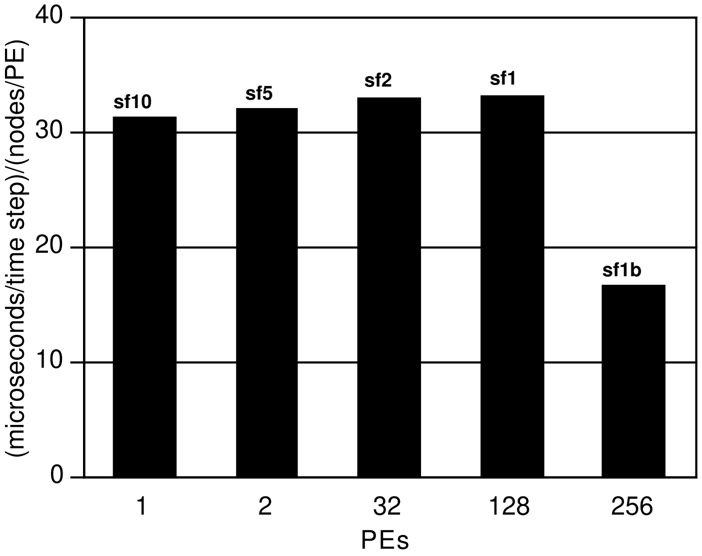

Figure 10: T3D wall-clock time in microseconds per time step per average number of nodes per processor (PE), as a function of number of processors. This figure is based on an entire 6000 time step simulation, exclusive of I/O. The `sf1b` result is based on a damping scheme in which $\beta = 0$ in Equation 5 so that only one matrix-vector product is performed at each time step.

above, for a truly scalable algorithm/implementation/hardware system, this number should remain constant as problem size increases with increasing processors. The figure demonstrates that we are close to this ideal. Ultimately, wall clock time per node per time step is the most meaningful measure of scalable performance for our application, since it is a direct indicator of the ability to solve our ultimate target problems, which are an order of magnitude larger than the San Fernando Basin problem we have described in this paper.

6 Concluding remarks

We have described our approach to modeling the earthquake-induced ground motion in large, heterogeneous basins on parallel computers. By paying careful attention to the impact on parallel execution of all components of the code, we are able to obtain excellent performance on highly unstructured mesh problems. In particular, through the use of (i) space- and time-localized absorbing boundaries; (ii) seismic input in the form of effective boundary or interior forces applied at the element level; (iii) explicit numerical techniques for the wave propagation problem; (iv) strict control of mesh resolution and aspect ratio; and (v) an asymptotically optimal mesh partitioner, we obtain excellent scalability of the parallel code. The Archimedes toolset integrates the basic components necessary for solving general PDE problems involving static unstructured meshes on parallel distributed memory systems. These components include meshing, partitioning, and parallel code generation. Archimedes has been instrumental in allowing us to quickly build and test parallel ground motion simulation codes.

We currently solve the meshing, partitioning, and parceling problems sequentially on a large shared-memory machine. Our ultimate target problem—the Greater Los Angeles Basin with an excitation of 2 Hz and with soil deposits having shear wave velocities as low as 200 m/s—will require meshes on the order of hundreds of millions of elements. Despite the fact that our sequential meshing and partitioning codes are fast, we may have to parallelize these steps in order to solve the target problem, primarily for memory reasons. The scalability of the parallel portion of our code indicates that our target problem is within reach.

7 Acknowledgments

This research was supported by the National Science Foundation's Grand Challenges in High Performance Computing and Communications program, under grant CMS-9318163. Funding comes from the Directorate for Computer and Information Science and Engineering, the Directorate for Engineering, and the Directorate for Earth and Atmospheric Sciences. In addition, NSF funding was supplemented with funds from the Advanced Research Projects Agency. Computing services on the Pittsburgh Supercomputing Center's Cray T3D and DEC 8400 were provided under PSC grant BCS-960001P. We thank Harold Magistrale and Steve Day of San Diego State University for providing the material property model of the San Fernando Valley.

References

[1] K. Aki. Local site effect on ground motion. In J. Lawrence Von Thun, editor, *Earthquake Engineering and Soil Dynamics. II: Recent Advances in Ground-Motion Evaluation*, pages 103–155. ASCE, 1988.

[2] `http://www.cs.cmu.edu/~quake/archimedes.html`.

[3] Jacobo Bielak and Paul Christiano. On the effective seismic input for nonlinear soil-structure interaction systems. *Earthquake Engineering and Structural Dynamics*, 12:107–119, 1984.

[4] Jacobo Bielak, Loukas F. Kallivokas, Jifeng Xu, and Richard Monopoli. Finite element absorbing boundary for the wave equation in a halfspace with an application to engineering seismology. In *Proceedings of the Third International Conference on the Mathematical and Numerical Aspects of Wave Propagation*, pages 489–498, Mandelieu-La Napoule, France, April 1995. SIAM and INRIA.

[5] Adrian Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[6] Marco G. Cremonini, Paul Christiano, and Jacobo Bielak. Implementation of effective seismic input for soil-structure interaction systems. *Earthquake Engineering and Structural Dynamics*, 16:615–625, 1988.

[7] Anja Feldmann, Omar Ghattas, John R. Gilbert, Gary L. Miller, David R. O'Hallaron, Eric J. Schwabe, Jonathan Richard Shewchuk, and Shang-Hua Teng. Automated parallel solution of unstructured PDE problems. To appear, 1996.

[8] Arthur Frankel and John E. 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*, 82:2045–2074, 1992.

[9] Robert W. Graves. Modeling three-dimensional site response effects in the Marina District Basin, San Francisco, California. *Bulletin of the Seismological Society of America*, 83:1042–1063, 1993.

[10] Loukas F. Kallivokas, Jacobo Bielak, and Richard C. MacCamy. Symmetric local absorbing boundaries in time and space. *Journal of Engineering Mechanics, ASCE*, 117:2027–2048, 1991.

[11] Hiroshi Kawase and Keiiti Aki. A study on the response of a soft basin for incident S, P, and Rayleigh waves with special reference to the long duration observed in Mexico City. *Bulletin of the Seismological Society of America*, 79:1361–1382, 1989.

[12] Hsui-Lin Liu and Thomas Heaton. Array analysis of the ground velocities and accelerations from the 1971 San Fernando, California, earthquake. *Bulletin of the Seismological Society of America*, 74:1951–1968, 1996.

[13] Harold Magistrale, Keith L. McLaughlin, and Steven M. Day. A geology-based 3-D velocity model of the Los Angeles basin sediments. Submitted to *Bulletin of the Seismological Society of America*, 1996.

[14] Keith L. McLaughlin and Steven M. Day. 3D elastic finite difference seismic wave simulations. *Computers in Physics*, Nov/Dec 1994.

[15] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Automatic mesh partitioning. In Alan George, John Gilbert, and Joseph Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *The IMA Volumes in Mathematics and its Application*, pages 57–84. Springer-Verlag, 1993.

[16] Kim B. Olsen and Ralph J. Archuleta. Three-dimensional simulation of earthquakes on the Los Angeles Fault System. *Bulletin of the Seismological Society of America*, 86:575–596, 1996.

[17] Kim B. Olsen, Ralph J. Archuleta, and Joseph R. Matarese. Magnitude 7.75 earthquake on the San Andreas fault: Three-dimensional ground motion in Los Angeles. *Science*, 270(5242):1628–1632, 1995.

[18] Francisco J. Sanchez-Sesma and Francisco Luzon. Seismic response of three-dimensional valleys for incident P, S, and Rayleigh waves. *Bulletin of the Seismological Society of America*, 85:269–284, 1995.

[19] Jonathan Richard Shewchuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. Association for Computing Machinery, May 1996.

[20] Jonathan Richard Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133. Association for Computing Machinery, May 1996.

[21] Jonathan Richard Shewchuk and Omar Ghattas. A compiler for parallel finite element methods with domain-decomposed unstructured meshes. In David E. Keyes and Jinchao Xu, editors, *Domain Decomposition Methods in Scientific and Engineering Computing*, volume 180 of *Contemporary Mathematics*, pages 445–450. American Mathematical Society, 1994.

[22] John E. Vidale and Donald V. Helmberger. Elastic finite-difference modeling of the 1971 San Fernando, California, earthquake. *Bulletin of the Seismological Society of America*, 78:122–141, 1988.

[23] David F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoï polytopes. *Computer Journal*, 24(2):167–172, 1981.