SIAM J. SCI. COMPUT.
Vol. 24, No. 1, pp. 1–19

SPATIAL PARALLELISM OF A 3D FINITE DIFFERENCE VELOCITY-STRESS ELASTIC WAVE PROPAGATION CODE*

SUSAN E. MINKOFF†

Abstract. In a three-dimensional isotropic elastic earth, the wave equation solution consists of three velocity components and six stresses. We discretize the partial derivatives using second order in time and fourth order in space staggered finite difference operators. The parallel implementation uses the message passing interface library for platform portability and a spatial decomposition for efficiency. Most of the communication in the code consists of passing subdomain face information to neighboring processors. When the parallel communication is balanced against computation by allocating subdomains of reasonable size, we observe excellent scaled speedup. Allocating subdomains of size $25 \times 25 \times 25$ on each node, we achieve efficiencies of 94% on 128 processors of an Intel Paragon.

Key words. seismic-wave propagation, finite difference methods, parallel computing, elastic-wave theory, reflection seismology

AMS subject classifications. 86–08, 86A15, 65Y05

PII. S1064827501390960

1. Introduction.

Realistic-sized elastic wave propagation simulations in three dimensions are extremely computationally intensive. Even assuming an isotropic elastic earth (which reduces the number of unknowns and equations to nine), reasonable-sized simulations may require too much memory and too much time to run on a single-processor machine. Further, finite difference schemes for solving the wave equation, although very expensive, are able to model more of the physics of wave propagation than can be achieved using approximate (e.g., asymptotic) schemes. Attempts to improve on memory management for single-processor simulations include such finite difference variants as core memory optimization for staggered schemes [5], core and disk memory optimization [11], and the use of staggered grids with nonuniform spacing [14]. Ultimately, however, parallel computation is one of the most efficient solutions for running finite difference simulations in three dimensions.

The original papers on explicit staggered finite difference schemes were written by Madariaga [10] and Virieux [17], [18]. Emerman, Schmidt, and Stephen [4] give an interesting discussion of using an implicit scheme to solve these equations. Their work indicates that explicit schemes are preferable for the wave equation. More recently, a number of papers have appeared which attempt to improve on or analyze aspects of the staggered algorithm. Some examples include the work of Bayliss et al. [3], Levander [8], Luo and Schuster [9], Graves [5], Moczo et al. [11], and Pitarka [14]. A dispersion analysis of a staggered displacement-stress formulation is given by Sei [15]. Work on parallel, staggered finite difference schemes for the elastic wave equation include the earthquake modeling paper of Olsen, Archuleta, and Matarese [12], and the work of Hestholm [6] and Hestholm and Ruud [7], who use higher-order

*Received by the editors June 18, 2001; accepted for publication (in revised form) January 4, 2002; published electronically May 20, 2002. This research was supported by Sandia National Labs under contract DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company.

http://www.siam.org/journals/sisc/24-1/39096.html

†Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250 (sminkoff@math.umbc.edu).

differencing and include research on coordinate change of variables for uneven surface topography.

In this work we apply spatial (or data) parallelism via the message passing interface (MPI) to the three-dimensional (3D) isotropic elastic wave equations which have been discretized via second order in time and fourth order in space staggered finite difference operators. The serial code is perhaps most unusual in its accurate modeling of multiple types of seismic sources. Data parallelism is the most efficient way to parallelize finite difference simulations. On a massively parallel (mp) machine, our code is able to achieve a scaled speedup of 94% on 128 nodes where (for example) each processor solves a small subproblem of size $25 \times 25 \times 25$. Because the bulk of the cost of this simulation occurs in the time-step loop, the one-time reads of the earth model and writes of seismograms and plane slices are handled by processor 0 issuing *broadcasts* and *reduces* for data transmission. To allow the user to place sources, receivers, and requests for slice output anywhere in the physical domain (rather than only at grid point locations), cubic interpolation and extrapolation are used extensively in the serial code. In the parallel code, each processor accurately computes its finite difference updates, as well as these extrapolated values, by ensuring its neighboring processors have four planes of up-to-date ghost-cell information for each subdomain face (required for extrapolation of edge points). Within the time-step loop all communication is accomplished via basic (linear-cost) *sends* and *receives*. The parallel code runs on one-dimensional (1D), two-dimensional (2D), or 3D processor decompositions (with varying levels of efficiency).

In the remainder of the paper we describe the elastic wave equations we solve and the staggered finite difference scheme. We give a description of spatial parallelism in general and its use in this code. We also describe how i/o is handled and present theoretical cost estimates for the finite difference update algorithm and numerical timing studies for both fixed- and scaled-size problems. Finally, we describe two numerical examples. The first example simulates waves in a simple layered-earth model. The parallel code results match single-node results to full accuracy as illustrated in this paper with plane-slice output. The second example is larger and consists of a homogeneous earth model with a high-velocity blocky inclusion in the top corner of the domain. This example indicates that the parallel code is indeed capable of capturing 3D wave propagation in a heterogeneous earth.

2. The serial algorithm.

2.1. Governing equations. The serial algorithm description in this section is taken in abbreviated form from Chapter 4 of the technical report by Sleefe et al. [16]. For a more classical reference to elastic wave propagation see [1]. We solve the elastic wave equation in a 3D medium occupying a volume V and with boundary S. The medium may be described by Lamé parameters $\lambda(\vec{x})$ and $\mu(\vec{x})$ and mass density $\rho(\vec{x})$, where $\vec{x} \in \Re^3$. If $\alpha(\vec{x})$, $\beta(\vec{x})$ are the compressional (P) and shear (S) wave speeds, respectively, then we can relate the wave speeds and Lamé parameters via

$$\lambda(\vec{x}) = \rho(\vec{x})[\alpha(\vec{x})^2 - 2\beta(\vec{x})^2] \quad \text{and} \quad \mu(\vec{x}) = \rho(\vec{x})\beta(\vec{x})^2.$$

The velocity-stress form of the elastic wave equation consists of nine coupled, first-order partial differential equations for the three particle velocity vector components $v_i(\vec{x}, t)$ and the six independent stress tensor components $\sigma_{ij}(\vec{x}, t)$. Here $i, j = 1, 2, 3$

and the stress tensor is isotropic; so $\sigma_{ij} = \sigma_{ji}$. The 3D velocity-stress equations are

$$\frac{\partial v_i(\vec{x},t)}{\partial t} - b(\vec{x})\frac{\partial \sigma_{ij}(\vec{x},t)}{\partial x_j} = b(\vec{x})\left[f_i(\vec{x},t) + \frac{\partial m^a_{ij}(\vec{x},t)}{\partial x_j}\right],$$

$$\frac{\partial \sigma ij(\vec{x},t)}{\partial t} - \lambda(\vec{x})\frac{\partial v_k(\vec{x},t)}{\partial x_k}\delta_{ij} - \mu(\vec{x})\left[\frac{\partial v_i(\vec{x},t)}{\partial x_j} + \frac{\partial v_j(\vec{x},t)}{\partial x_i}\right] = \frac{\partial m^s_{ij}(\vec{x},t)}{\partial t}.$$

Here, $b = 1/\rho$ is the mass buoyancy. Sources of seismic waves are denoted by f_i (force source) or m_{ij} (moment source). The symmetric and antisymmetric parts of the moment density tensor are denoted (respectively) by

$$m^s_{ij}(\vec{x},t) = 1/2[m_{ij}(\vec{x},t) + m_{ji}(\vec{x},t)], \quad m^a_{ij}(\vec{x},t) = 1/2[m_{ij}(\vec{x},t) - m_{ji}(\vec{x},t)].$$

The traction boundary condition (or normal component of stress) must satisfy

$$\sigma_{ij}(\vec{x},t)n_j(\vec{x}) = t_i(\vec{x},t)$$

for $\vec{x}$ on S, where $t_i(\vec{x},t)$ are the components of the time-varying surface traction vector and $n_i(\vec{x})$ are the components of the outward unit normal to S. Initial conditions on the dependent variables are specified throughout V and on S at time $t = t_0$ via

$$v_i(\vec{x},t_0) = v^0_i(\vec{x}), \quad \sigma_{ij}(\vec{x},t_0) = \sigma^0_{ij}(\vec{x}).$$

2.2. Sources and seismograms. Numerous source types are handled by the code. For instance, a unidirectional point force acting in a volume V is represented by the force density vector

$$f_i(\vec{x},t) = Fw(t)d_i\delta(\vec{x} - \vec{x_s}),$$

where F is a force amplitude scalar, $w(t)$ is a dimensionless source waveform, and d_i are the components of a dimensionless unit vector giving the orientation of the applied force.

As a second example, a point moment acting within V is described by the moment density tensor

$$m_{ij}(\vec{x},t) = -Mw(t)d_{ij}\delta(\vec{x} - \vec{x_s})$$

with M a moment amplitude scalar, $w(t)$ a dimensionless source waveform, and d_{ij} are the components of a second-rank tensor giving the orientation of the applied moment. Seismic sources involving force dipoles, force couples, torques, and explosions/implosions are represented by a moment density tensor.

On output, the code produces both seismograms and 2D plane slices. The seismic traces represent particle velocity or acoustic pressure. If the orientation of the transducer sensitivity axis is defined by the dimensionless unit vector $\vec{b}$, then the particle velocity seismogram is

$$v_b(\vec{x_r},t) = b_k v_k(\vec{x_r},t) = b_1 v_1(\vec{x_r},t) + b_2 v_2(\vec{x_r},t) + b_3 v_3(\vec{x_r},t)$$

and the acoustic pressure seismogram is

$$p(\vec{x_r},t) = -\frac{1}{3}\sigma_{kk}(\vec{x_r},t) = -\frac{1}{3}[\sigma_{11}(\vec{x_r},t) + \sigma_{22}(\vec{x_r},t) + \sigma_{33}(\vec{x_r},t)].$$

2.3. Staggered finite difference scheme. The finite difference algorithm is an explicit scheme which is second-order accurate in time and fourth-order accurate in space. Staggered grid storage allows the partial derivatives to be approximated by centered finite differences without doubling the spatial extent of the operators, thus providing more accuracy (see [8]). Here we apply the finite difference scheme to the above equations after they have been nondimensionalized. For a discussion of the motivation and manner of equation nondimensionalization see Chapter 4 of [16].

We choose a discretization of the 3D spatial grid so that $x_i = x_0 + (i-1)h_x$, $y_j = y_0 + (j-1)h_y$, and $z_k = z_0 + (k-1)h_z$ for $i = 1, 2, 3, \ldots, I$, $j = 1, 2, 3, \ldots, J$, and $k = 1, 2, 3, \ldots, K$, respectively. Here x_0, y_0, z_0 are the minimum grid values and h_x, h_y, h_z give the distance between grid points in the three coordinate directions. Values of the two Lamé parameters and mass buoyancy at a node are given by $\lambda(x_i, y_j, z_k)$, $\mu(x_i, y_j, z_k)$, and $b(x_i, y_j, z_k)$. The time discretization is defined by $t_l = t_0 + (l-1)h_t$ for $l = 1, 2, 3, \ldots, L$. Here t_0 is the minimum time and h_t is the time increment.

Subsets of the dependent variables are stored on different spatial and temporal grids. Specifically, the diagonal components of the stress tensor are stored on the grid points defined above:

$$\sigma_{xx}(x_i, y_j, z_k, t_l), \quad \sigma_{yy}(x_i, y_j, z_k, t_l), \quad \sigma_{zz}(x_i, y_j, z_k, t_l).$$

The off-diagonal stress tensor components are shifted in space by one-half grid interval along two coordinate axes:

$$\sigma_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l), \quad \sigma_{yz}(x_i, y_j + h_y/2, z_k + h_z/2, t_l),$$
$$\sigma_{xz}(x_i + h_x/2, y_j, z_k + h_z/2, t_l).$$

Finally, the three velocity vector components are stored at grid points that are shifted in both space and time by one-half grid interval:

$$v_x(x_i + h_x/2, y_j, z_k, t_l + h_l/2), \quad v_y(x_i, y_j + h_y/2, z_k, t_l + h_t/2),$$
$$v_z(x_i, y_j, z_k + h_z/2, t_l + h_t/2).$$

We can define (for example) the following oft-used (dimensionless) quantities:

$$p_x = \frac{c_1 h_t S_w}{h_x}, \quad p_y = \frac{c_1 h_t S_w}{h_y}, \quad p_z = \frac{c_1 h_t S_w}{h_z}, \quad q_x = \frac{c_2 h_t S_w}{h_x},$$
$$q_y = \frac{c_2 h_t S_w}{h_y}, \quad q_z = \frac{c_2 h_t S_w}{h_z},$$

where S_w is the characteristic unit of measure for wavespeed. Similarly, we can define S_ρ, S_v, and S_σ as characteristic units of measure for the mass density, velocity vector, and stress tensor components, respectively. As an example, one might choose $S_w = 10^3$ m/s and $S_\rho = 10^3$ kg/m^3 as appropriate normalizing values. We generally choose $c_1 = 9/8$ and $c_2 = -1/24$ as coefficients in the fourth-order finite difference operator.

Then the time update formulas for the three representative unknowns (velocity, diagonal stress component, and off-diagonal stress component) are given in (2.1), (2.2), (2.3). The other two velocity and four stress unknowns are similar in form to

these examples. We solve the following finite difference equation for the x-component of the particle velocity vector:

(2.1)
$$\begin{aligned}
v_x(x_i + h_x/2, y_j, z_k, t_l + h_t/2) &= v_x(x_i + h_x/2, y_j, z_k, t_l - h_t/2) \\
&+ b(x_i + h_x/2, y_j, z_k)\{p_x\left[\sigma_{xx}(x_i + h_x, y_j, z_k, t_l) - \sigma_{xx}(x_i, y_j, z_k, t_l)\right] \\
&+ q_x\left[\sigma_{xx}(x_i + 2h_x, y_j, z_k, t_l) - \sigma_{xx}(x_i - h_x, y_j, z_k, t_l)\right] \\
&+ p_y\left[\sigma_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l) - \sigma_{xy}(x_i + h_x/2, y_j - h_y/2, z_k, t_l)\right] \\
&+ q_y\left[\sigma_{xy}(x_i + h_x/2, y_j + 3h_y/2, z_k, t_l) - \sigma_{xy}(x_i + h_x/2, y_j - 3h_y/2, z_k, t_l)\right] \\
&+ p_z\left[\sigma_{xz}(x_i + h_x/2, y_j, z_k + h_z/2, t_l) - \sigma_{xz}(x_i + h_x/2, y_j, z_k - h_z/2, t_l)\right] \\
&+ q_z\left[\sigma_{xz}(x_i + h_x/2, y_j, z_k + 3h_z/2, t_l) - \sigma_{xz}(x_i + h_x/2, y_j, z_k - 3h_y/2, t_l)\right]\} \\
&+ b(x_i + h_x/2, y_j, z_k)\{\tfrac{h_t}{S_\rho S_v} f_x(x_i + h_x/2, y_j, z_k, t_l) \\
&+ p_y\left[m^a_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l) - m^a_{xy}(x_i + h_x/2, y_j - h_y/2, z_k, t_l)\right] \\
&+ q_y\left[m^a_{xy}(x_i + h_x/2, y_j + 3h_y/2, z_k, t_l) - m^a_{xy}(x_i + h_x/2, y_j - 3h_y/2, z_k, t_l)\right] \\
&+ p_z\left[m^a_{xz}(x_i + h_x/2, y_j, z_k + h_z/2, t_l) - m^a_{xz}(x_i + h_x/2, y_j, z_k - h_z/2, t_l)\right] \\
&+ q_z\left[m^a_{xz}(x_i + h_x/2, y_j, z_k + 3h_z/2, t_l) - m^a_{xz}(x_i + h_x/2, y_j, z_k - 3h_z/2, t_l)\right].
\end{aligned}$$

In the formula above, the buoyancy (b) between grid nodes is derived from an appropriate order of interpolation. The xx-component of the stress tensor may be solved for from the following equation:

(2.2)
$$\begin{aligned}
\sigma_{xx}(x_i, y_j, z_k, t_l + h_t) &= \sigma_{xx}(x_i, y_j, z_k, t_l) + \left[\lambda(x_i, y_j, z_k) + 2\mu(x_i, y_j, z_k)\right] \\
&\times \{p_x\left[v_x(x_i + h_x/2, y_j, z_k, t_l + h_t/2) - v_x(x_i - h_x/2, y_j, z_k, t_l + h_t/2)\right] \\
&+\ q_x\left[v_x(x_i + 3h_x/2, y_j, z_k, t_l + h_t/2) - v_x(x_i - 3h_x/2, y_j, z_k, t_l + h_t/2)\right]\} \\
&+ \lambda(x_i, y_j, z_k)\{p_y\left[v_y(x_i, y_j + h_y/2, z_k, t_l + h_t/2) - v_y(x_i, y_j - h_y/2, z_k, t_l + h_t/2)\right] \\
&+ q_y\left[v_y(x_i, y_j + 3h_y/2, z_k, t_l + h_t/2) - v_y(x_i, y_j - 3h_y/2, z_k, t_l + h_t/2)\right] \\
&+ p_z\left[v_z(x_i, y_j, z_k + h_z/2, t_l + h_t/2) - v_z(x_i, y_j, z_k - h_z/2, t_l + h_t/2)\right] \\
&+\ q_z\left[v_z(x_i, y_j, z_k + 3h_z/2, t_l + h_t/2) - v_z(x_i, y_j, z_k - 3h_z/2, t_l + h_t/2)\right]\} \\
&+\ \left[m^s_{xx}(x_i, y_j, z_k, t_l + h_t) - m^s_{xx}(x_i, y_j, z_k, t_l)\right].
\end{aligned}$$

No interpolation for Lamé parameters is needed in the formula above. Finally, one solves for the xy-component of the stress tensor via

(2.3)
$$\begin{aligned}
\sigma_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l + h_t) &= \sigma_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l) \\
&+ \mu(x_i + h_x/2, y_j + h_y/2, z_k) \\
&\times \{p_x\left[v_y(x_i + h_x, y_j + h_y/2, z_k, t_l + h_t/2) - v_y(x_i, y_j + h_y/2, z_k, t_l + h_t/2)\right] \\
&+ q_x\left[v_y(x_i + 2h_x, y_j + h_y/2, z_k, t_l + h_t/2) - v_y(x_i - h_x, y_j + h_y/2, z_k, t_l + h_t/2)\right] \\
&+ p_y\left[v_x(x_i + h_x/2, y_j + h_y, z_k, t_l + h_t/2) - v_x(x_i + h_x/2, y_j, z_k, t_l + h_t/2)\right] \\
&+\ q_y\left[v_x(x_i + h_x/2, y_j + 2h_y, z_k, t_l + h_t/2) - v_x(x_i + h_x/2, y_j - h_y, z_k, t_l + h_t/2)\right]\} \\
&+\ \left[m^s_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l + h_t) - m^s_{xy}(x_i + h_x/2, y_j + h_y/2, z_k, t_l)\right].
\end{aligned}$$

The shear modulus (μ) between grid nodes is again obtained by interpolation.

3. Spatial parallelism. The most efficient parallel programs are ones which attempt to minimize the communication between processors while still requiring each processor to accomplish basically the same amount of work. This trade-off between load balancing and the cost of communication is best achieved for an explicit finite difference scheme via spatial (data) parallelism. Spatial parallelism also tends to scale well (see [13, Chap. 10]). In this case, the original serial algorithm was not modified

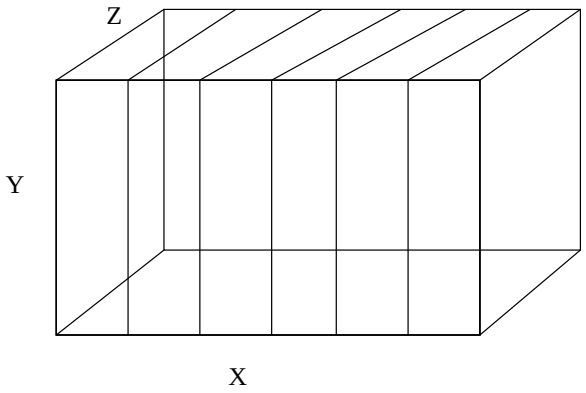

FIG. 1. 1*D processor decomposition (in the x direction) for spatial parallelism.*

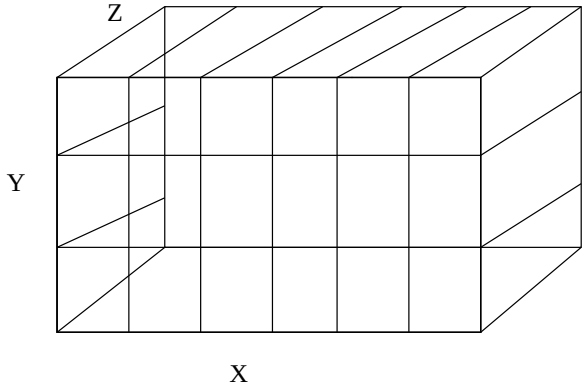

FIG. 2. 2*D processor decomposition (in the x and y directions) for spatial parallelism.*

to improve parallel efficiency, although quite good scaled speedup is achieved on mp machines.

In spatial parallelism, the physical problem domain is split among the processors so that each processor solves its own subdomain problem. In our implementation, the user may specify processor decompositions in one, two, or three dimensions, although generally a more balanced division (3D decomposition) is the most efficient. Further, there is no requirement that the number of processors in any one direction must evenly divide the number of grid points. Divisions with remainders are allowed and are transparent to the user. A typical 1D decomposition is illustrated in Figure 1, a 2D decomposition in Figure 2, and a 3D decomposition in Figure 3.

In order for each processor to completely calculate the finite difference solutions for its subdomain independent of the other processors, we allocate padded subdomains (ghost cells) of memory (see Figure 4 for an illustration of a processor subdomain). For a typical fourth-order spatial finite difference scheme, two extra planes of memory need to be allocated on each face of the subdomain cube. Because the serial algorithm allows the user the flexibility of specifying sources, receivers, and plane-slice output anywhere in the domain (not necessarily on a grid node), cubic interpolation and extrapolation are used extensively throughout the code. Therefore, we allocate four

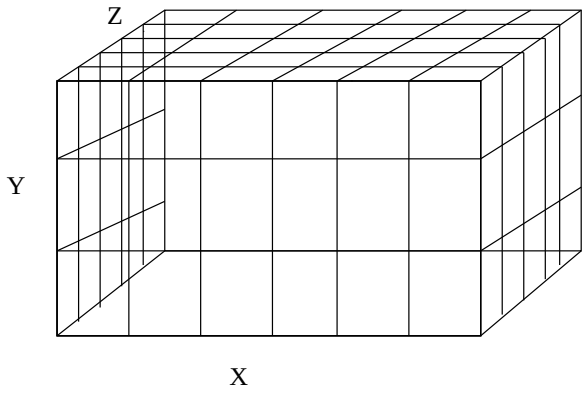

FIG. 3. *3D processor decomposition for spatial parallelism.*

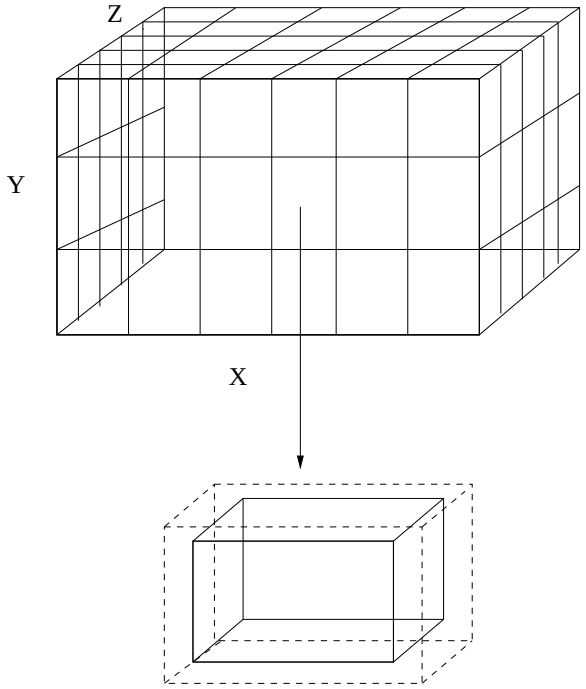

FIG. 4. *Single processor subdomain (with ghost cells shown as dashed lines) taken from global grid domain.*

planes of ghost cells for each face of the subdomain cube to ensure the extrapolation (even for boundary points) has the correct data values at each time step. Most of the subroutines do not need to communicate all four planes of data from processor to processor, however.

After establishing the correct Cartesian communicator for the processor decomposition, we use the MPI *shift* commands to ensure that every processor sends edge information to its neighboring processors (up, down, right, left, front, back). Every subroutine which updates grid values of velocity or stress must send this information

at the start of the routine. The affected routines include those which do the actual finite difference updates, ones which calculate the absorbing boundary conditions or insert sources, and all the extrapolation routines (including the output seismogram and plane-slice routines). This rather sizeable amount of communication negatively impacts parallel efficiency when the size of the subdomain problems is small relative to the number of the ghost cells. However, for reasonable-sized subdomain problems, the scaled speedup is still quite good (see Figure 7 and Table 1).

4. I/O issues. For realistic-sized domains, the majority of the time spent in this algorithm will occur in the time-step loop. Therefore, I did not focus on researching optimal parallel i/o strategies. Nonetheless, some simple measures were taken to improve efficiency. The algorithm reads input information from at least four different files. Most of the recording geometry data and basic run parameters are scalars which processor 0 reads in and then broadcasts to all other processors. These scalar values are packaged and broadcast with a single call. Reading the earth model information (velocities and density at each grid point) is the most costly of the i/o operations. Processor 0 reads in a line (set of records) from the external file and broadcasts this information to all processors. Each processor then determines whether its subdomain uses that section of the earth model and unpacks and stores the data it requires. This technique prevents processors from sitting idle during the initial reading of input data. Although this information is only read once, improvements to reading the earth model could undoubtedly be made.

The output routines (seismogram and plane slice) use MPI operators that are the inverse of the input routines. The plane-slice routine currently requires every processor to allocate the total memory for a plane slice. The processors fill out their portion of that plane and then a global sum over each point in the plane (*reduce* operation) is performed before processor 0 writes the output to an external file. If the user specifies a receiver or slice plane not on a grid node, cubic extrapolation is used. Therefore, it is possible that (even at a single grid point in the plane slice) only part of the weighted combination of output values will be stored on a certain processor. The remainder of values needed for the extrapolation might be on a neighboring processor. The *reduce* operation correctly handles receivers or slice planes which occur in the physical domain on boundaries between processor subgrids (see Figure 5 for an illustration).

The collective communication calls *broadcast* and *reduce* are more expensive than simple *sends* and *receives*, but these commands are used only in the i/o routines. All message passing used in the finite difference computation is accomplished with (linear cost) *sends/receives*.

5. Timing studies.

5.1. Background. Two of the most important concepts in parallel programming are computational speedup and parallel efficiency. Speedup is loosely defined by how much faster a program will run on n processors than on a single processor. This quantity can be measured in two basic ways. The first way is to fix the size of the problem solved and vary the number of processors (fixed-size problem). The second is to allow the number of processors used to solve the problem to grow commensurate with the growth in the problem size (scaled-size problem). Both metrics are important to consider, although scaled speedup appears to be more appropriate for the elastic wave propagation code as large problems (too big to fit on a single processor) were the initial motivation behind the parallelization effort.

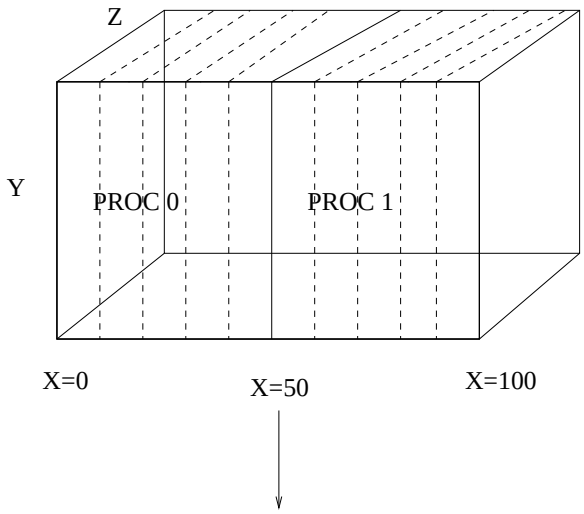

FIG. 5. *Example in which slice plane requested lies at the same point as the division of the grid between processors* 0 *and* 1.

For a fixed-size problem, *speedup* is defined by

$$S \equiv T_1(n)/T_p(n) \leq p,$$

where p = "ideal speedup" (see [2]) and $T_1(n)$, $T_p(n)$ are the times for running a problem of size n on 1 and p processors, respectively. *Efficiency* is defined

$$E \equiv T_1(n)/pT_p(n), \quad 0 \leq E \leq 1.$$

For a scaled-size problem, one generally must *estimate* the run time on a single processor. Speedup and efficiency are defined as follows:

$$S \equiv pT_1(n/p)/T_p(n), \quad E \equiv T_1(n/p)/T_p(n). \tag{5.1}$$

5.2. Theoretical cost. The cost estimate for this parallel algorithm is straightforward. I chose not to consider the cost of i/o in either the analysis or the numerical timing studies since the bulk of the work in this algorithm occurs in the propagation of waves (the wave equation update), not in reading the earth model or writing out seismograms. Thus, we must estimate the terms for computation and communication. Let $T(n,p) \equiv T_p(n)$. Then

$$T(n,p) = T_{comp}(n,p) + T_{comm}(n,p). \tag{5.2}$$

There are two important machine constants which impact the speed of message communication. The first is *latency*—the startup cost of sending a message (which is independent of message size). The second is *bandwidth* (message dependent) which is the (reciprocal of) transmission time/byte. Thus, if we denote machine latency by α and 1/bandwidth as β, the cost to send a single message with k units of data is $\alpha + k\beta$. On the Sandia Terraflops machine (an Intel Paragon), the following values

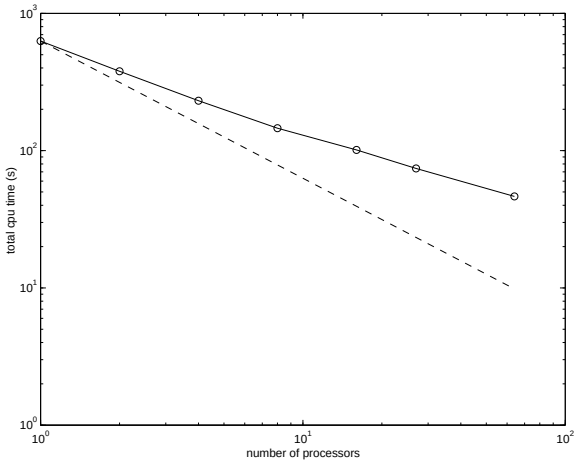

FIG. 6. *Timing curves for a fixed-size problem* $(48 \times 48 \times 48)$ *run on the Intel Paragon. The dashed curve is "ideal" or linear speedup, and the solid curve is the speedup exhibited by the parallel program. Circles further indicate data points.*

are approximately correct (in seconds): $\alpha = 2 \times 10^{-5}$, $\beta = 3 \times 10^{-9}$. Let τ denote the computation time per flop. On the Tflops machine, $\tau = 1 \times 10^{-8}$. (These timing constants are independently confirmed as reasonable for the Paragon in [13, pp. 251–252].)

If, for simplicity, we assume a uniform number of grid points in each direction, then the cost of performing a finite difference calculation on a grid of size $N \times N \times N$ on P processors is $C\tau N^3/P$, where the factor C takes into account the number of floating point operations in the finite difference stencil as well as the fact that there are nine output quantities to update (six stresses and three velocities). The fourth-order finite difference scheme requires us to send two plane faces of data to each neighbor in the 3D cube. (The cubic extrapolation *might* require an additional two plane faces be sent in each direction, for a total of four per direction.) So communication costs for a 1D decomposition are (maximally) $8(\alpha + 8\beta N^2)$. (Here the factor of 8 inside the parentheses comes from assuming each data point uses 8 bytes of memory.) For a 2D decomposition the cost of communication is $16(\alpha + 8\beta N^2/\sqrt{P})$. Finally, for a 3D decomposition we get $24(\alpha + 8\beta N^2/P^{2/3})$. (Note that we could replace the numbers 8, 16, 24 by another constant $\tilde{C}$, but we choose to explicitly specify these numbers in order to emphasize that communication cost depends on both the order of the finite difference stencil and the type of processor decomposition.) So, for a 3D decomposition,

$$T(N,P) = C\tau N^3/P + 24(\alpha + 8\beta N^2/P^{2/3}),$$

and speedup defined by $T(N,1)/T(N,P)$ is

$$C\tau N^3/\{C\tau N^3/P + 24(\alpha + 8\beta N^2/P^{2/3})\}.$$

Note that in our algorithm we do multiple transmissions of ghost-cell plane information not only during the finite difference update but also for insertion of sources at nongrid point locations, etc. So another constant could be inserted in front of the communication cost estimate for our algorithm (which would be problem dependent).

5.3. Numerical timing studies. Three timing studies will be discussed in this paper. The first is for a fixed-size problem with a total of $48 \times 48 \times 48$ grid points run on the Intel Paragon.

TABLE 1
Timings for scaled-size problem.

Scaled-size problem timings (without output) Each processor always solves problem of size $25 \times 25 \times 25$							
ProblemSize	P	P_x	P_y	P_z	Total run time (s)	Speedup	Efficiency
25x25x25	1	1	1	1	150.56	1.00	1.00
50x25x25	2	2	1	1	146.81	2.05	1.03
50x50x25	4	2	2	1	145.16	4.15	1.04
50x50x50	8	2	2	2	145.63	8.27	1.03
100x50x50	16	4	2	2	148.50	16.22	1.01
100x100x50	32	4	4	2	153.38	31.41	.98
100x100x100	64	4	4	4	160.36	60.09	.94
200x100x100	128	8	4	4	160.28	120.24	.94

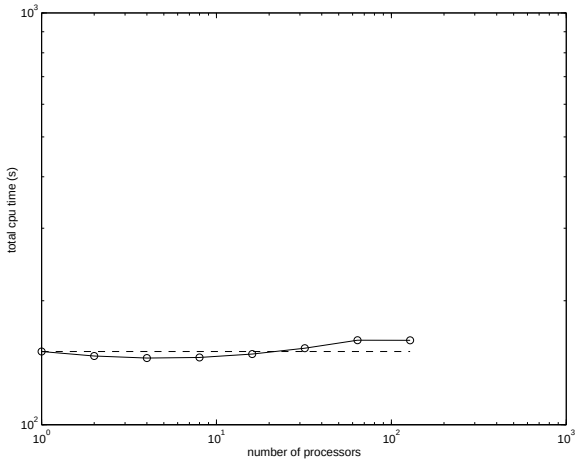

FIG. 7. *Timing curves for a scaled-size problem run on the Intel Paragon. Each node always solves problem of size* $25 \times 25 \times 25$. *Largest problem is size* $200 \times 100 \times 100$ *run on an* $8 \times 4 \times 4$ *processor grid. Dashed curve gives ideal speedup. Solid curve gives actual program speedup. Circles indicate data points.*

This problem was run on different numbers and configurations of processors ranging from $1 \times 1 \times 1$ to $4 \times 4 \times 4$ (total of 64 processors). 1D, 2D, and 3D decompositions were tested. In this case, communication costs were too high relative to the amount of computation performed, and efficiency dropped off rapidly (see Figure 6). A problem larger than about $50 \times 50 \times 50$ would not fit on a single node of our Intel Paragon.

The scaled-size problem showed much better speedup and efficiency (when also run on the Paragon). In this case a constant $25 \times 25 \times 25$ size problem was run on each node. From 1 to 128 nodes were used where the problem size on 128 nodes (processor decomposition of $8 \times 4 \times 4$) was $200 \times 100 \times 100$ grid points. Table 1 and Figure 7 illustrate the timings. Note that the efficiencies remain high (94%) even for 128 processors. In this paper no slice plane or seismogram output was produced for the timing study runs in an attempt to keep the algorithm simple (linear). More complicated computational cost is incurred with seismogram or slice output. In this case, we expect ideal speedup to be the straight line (constant) shown in Figure 7. In fact, timings were also done for runs which included seismogram and slice-plane output. Very little additional cost was incurred. However, the calculation of the

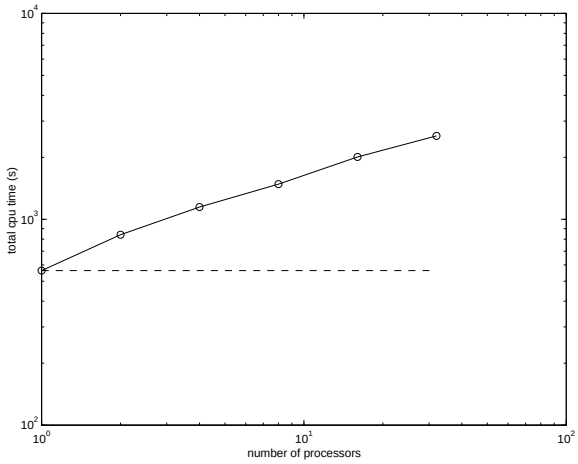

FIG. 8. *Timing curves for a scaled-size problem run on the Linux cluster. Each node always solves problem of size* $50 \times 50 \times 50$. *Largest problem is size* $200 \times 200 \times 100$ *run on an* $4 \times 4 \times 2$ *processor grid. Dashed curve gives ideal speedup. Solid curve gives actual program speedup. Circles indicate data points.*

communication portion of the theoretical cost function would need to include *reduce* ($O(\log(p))$ operations).

Finally, for comparison, we show a graph of scaled speedup timings for a homogeneous medium problem run on a *Linux cluster*. Since the cost of building clusters is currently so attractive when compared to the cost of building large mp machines, such a comparison seems worthwhile. Bandwidth and latency speeds on clusters tend to run 1–3 orders of magnitude slower than on the Paragon (an mp machine). For example, latency is typically about 2×10^{-5} s on the Paragon but 2×10^{-4} s on this Linux cluster. (In fact, latencies can often run to one millisecond for typical optimized clusters.) Reasonable numbers for reciprocal bandwidth (in seconds) are 3×10^{-9} for the Paragon and 1×10^{-8} for the cluster. (Linux cluster latency and bandwidth numbers are courtesy of O'Connor, relayed in a personal communication.) The communication costs on the cluster cause a drop in efficiency to 65% when going from a $50 \times 50 \times 50$ sized problem on a single node to running a $100 \times 50 \times 50$ sized problem on two nodes. The largest problem run in this study was size $200 \times 200 \times 100$ run on a $4 \times 4 \times 2$ processor decomposition. Figure 8 shows a graph of speedup for this problem. As in the fixed-size problem, the large amount of ghost-cell face communication due to extrapolation and interpolation of sources and receivers to grid nodes prohibits ideal speedup when high latency is present. In the absence of restructuring the serial algorithm (i.e., eliminating the excessive interpolation), such speedup is to be expected on a cluster.

6. Numerical examples. In this section, I describe two numerical examples to illustrate the parallel elastic wave propagation code. Although I could show a much larger real data example to test the limits of the parallel algorithm, I felt it was more important to use the examples to illustrate code accuracy and efficiency (timings). Example 1 consists of a small domain with $48 \times 48 \times 48$ grid points in (x, y, z). The grid spacing is 4 m in each direction. In this example the earth is assumed layered (velocities and density depend only on depth, z). This model has eight layers with P-wave velocity, S-wave velocity, and density ranging in value as shown in Table 2. The model is characterized by a shallow low-velocity layer near the top of the model and

TABLE 2
Earth model specifications for numerical example 1.

Layered earth example elastic properties			
Layer depth (m)	V_p (m/s)	V_s (m/s)	ρ (kg/m^3)
12	2500	1500	2000
20	1900	1300	2025
28	2000	1350	2025
40	2700	1600	2100
52	3000	1750	2100
76	3100	1780	2100
108	3300	1900	2227
196	3700	2000	2300

then a monotonic increase in velocity and density with depth below the low-velocity zone.

This small problem was run on both a single processor and on a processor decomposition of size 24 ($p_x = 2$, $p_y = 4$, $p_z = 3$) for 400 time steps (or 0.2 seconds total runtime). The Ricker source wavelet extends from $-.04$ s to .04 s with a peak frequency of 30 Hz. 22 receivers were located at varying x locations and $y = 80$ m. Plane slices of acoustic pressure in both the xy and yz planes ($z = 90$ m and $x = 90$ m, respectively) are shown in Figures 9 and 11 for the problem run on a single processor of the Intel Paragon. The same experiment was run on 24 processors, and the corresponding plane slices are shown in Figures 10 and 12. The wave field is spherical in the xy slices because the earth model varies only in the z direction. The yz slices are not spherical due to the variation in medium parameters. This example illustrates the accuracy of the parallel version of the code. In fact, the slices and seismograms are identical (to full precision) for the serial and parallel runs.

The second example is a larger case (with $150 \times 150 \times 150$ grid points) which demonstrates using the parallel code for simulating wave propagation in nonlayered media. The model's physical domain is size (750 m, 750 m, 750 m) with 5 m grid spacing in all directions. The earth model (shown in Figure 13) has homogeneous background velocities $V_p = 2500$ m/s, $V_s = 1500$ m/s, and density $\rho = 2000$ kg/m^3. Located at $(x, y, z) = (200$ m, 200 m, 200 m) (and parallel to the coordinate axes) is a fast (high-velocity) blocky inclusion of size (100 m, 100 m, 100 m). It has elastic properties $V_p = 4500$ m/s, $V_s = 2500$ m/s, and $\rho = 2200$ kg/m^3.

This larger problem would not fit on a single node of the Intel Paragon. It was run instead on 125 nodes (processor decomposition of $p_x = 5$, $p_y = 5$, $p_z = 5$) for 1 second of simulation time.

Figure 14 shows acoustic pressure snapshots of the yz plane slice through the blocky inclusion simulation at fixed x location $x = 250$ m and times 0, 0.05, 0.1, 0.15 seconds wave propagation. The first two snapshots are not affected by the high-velocity zone. The third and fourth, however, show the effect the inclusion has on the wave propagation.

Figures 15 and 16 give snapshots in the xy plane at fixed z location $z = 250$ m and times of 0, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15 seconds wave propagation. Again, the impact of the high-velocity zone is clearly seen in the second snapshot.

7. Conclusions. Finite difference solutions of the 3D elastic wave equation are considerably more accurate than asymptotic methods such as ray tracing. However, they are also much more expensive computationally. Two methods of speeding up

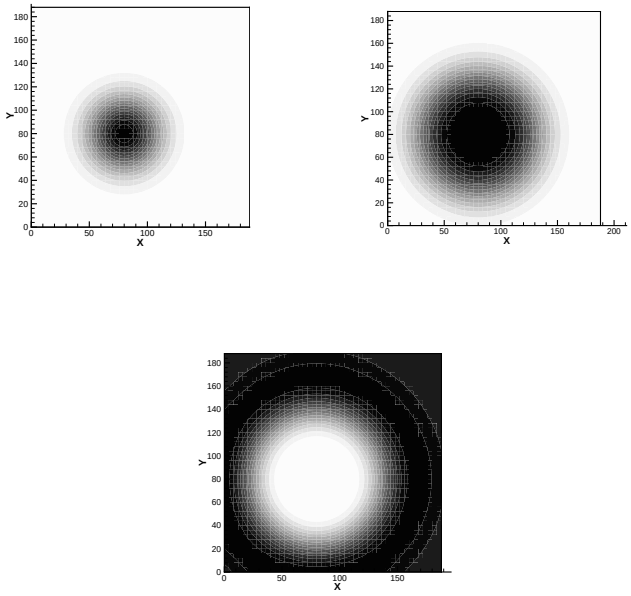

FIG. 9. *Plane slices in xy at depth* $z = 90$ *m for a problem of size* $48 \times 48 \times 48$ *run on a single processor. Figures in upper left, upper right, and lower middle correspond to pressure snapshots at times* 0, .0125, .025 *seconds wave propagation. This problem assumed a horizontally layered velocity and density model.*

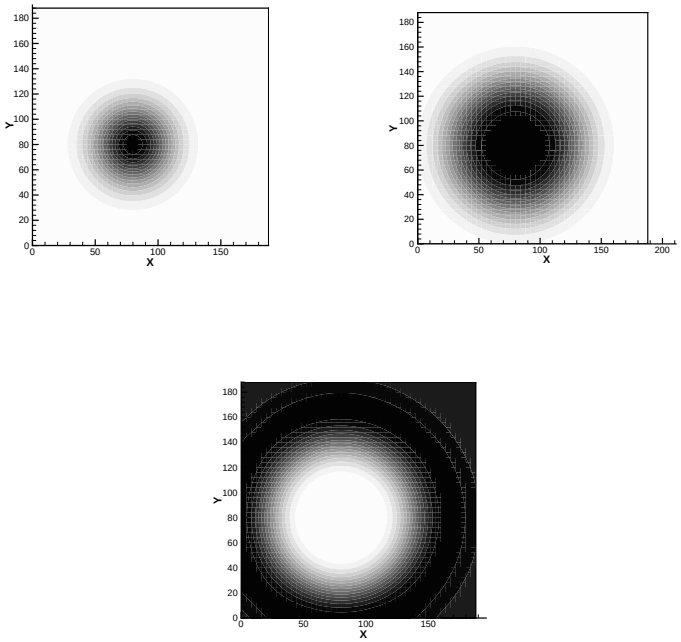

FIG. 10. *Plane slices in xy at depth* $z = 90$ *m for a problem of size* $48 \times 48 \times 48$ *run on a* $2 \times 4 \times 3$ *processor grid. Figures in upper left, upper right, and lower middle correspond to pressure snapshots at times* 0, .0125, .025 *seconds wave propagation. This problem assumed a horizontally layered velocity and density model.*

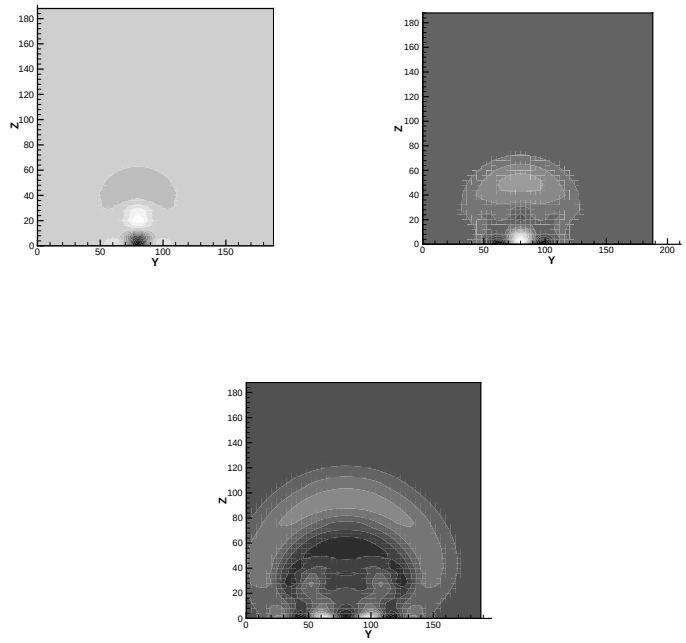

FIG. 11. *Plane slices in yz at fixed x location $x = 90$ m for a problem of size $48 \times 48 \times 48$ run on a single processor. Figures in upper left, upper right, and lower middle correspond to pressure snapshots at times* 0, .0125, .025 *seconds wave propagation. This problem assumed a horizontally layered velocity and density model.*

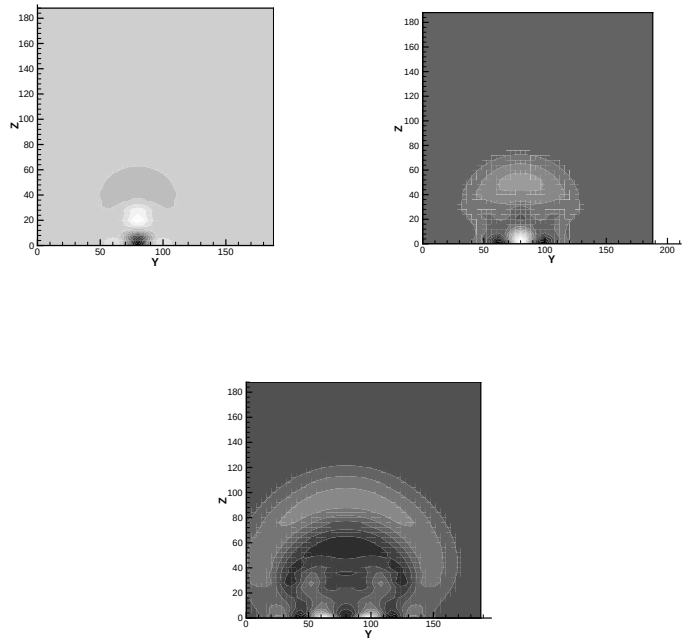

FIG. 12. *Plane slices in yz at fixed x location $x = 90$ m for a problem of size $48 \times 48 \times 48$ run on a $2 \times 4 \times 3$ processor grid. Figures in upper left, upper right, and lower middle correspond to pressure to snapshots at times* 0, .0125, .025 *seconds wave propagation. This problem assumed a horizontally layered velocity and density model.*

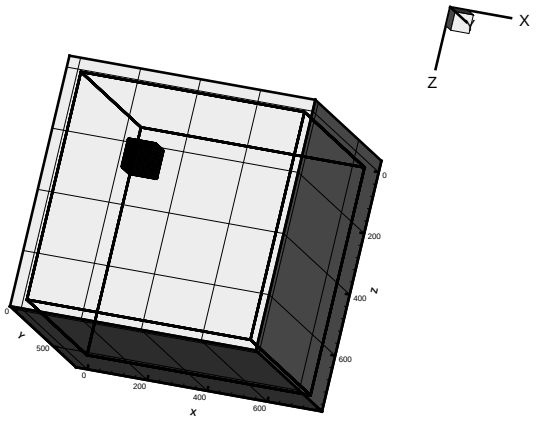

FIG. 13. *Earth model for experiment* 2. *Homogeneous medium with high-velocity blocky inclusion located at* $(x, y, z) = (200\ m,\ 200\ m,\ 200\ m)$.

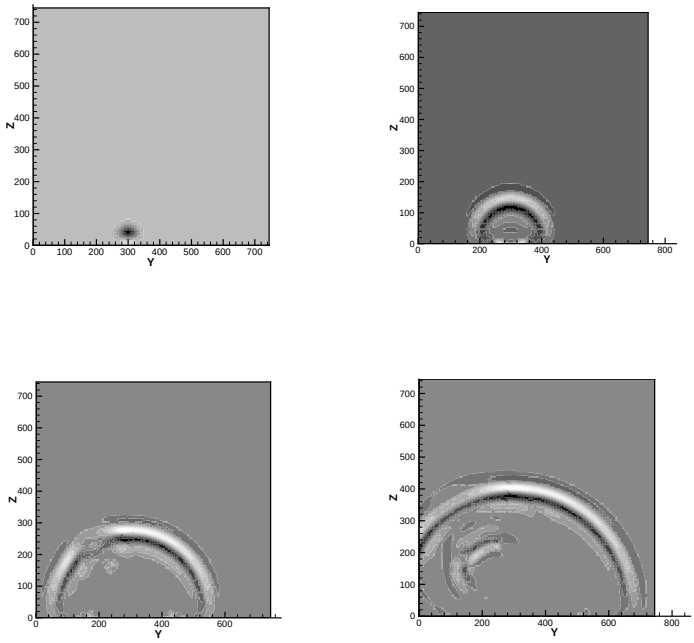

FIG. 14. *Plane slices in* yz *at fixed* $x = 250\ m$ *for the blocky inclusion problem run on a* $5 \times 5 \times 5$ *processor grid. Figures correspond to pressure snapshots at times* $0, 0.05,\ 0.1,\ 0.15$ *seconds. This problem had a blocky high-velocity cube embedded in a homogeneous background model.*

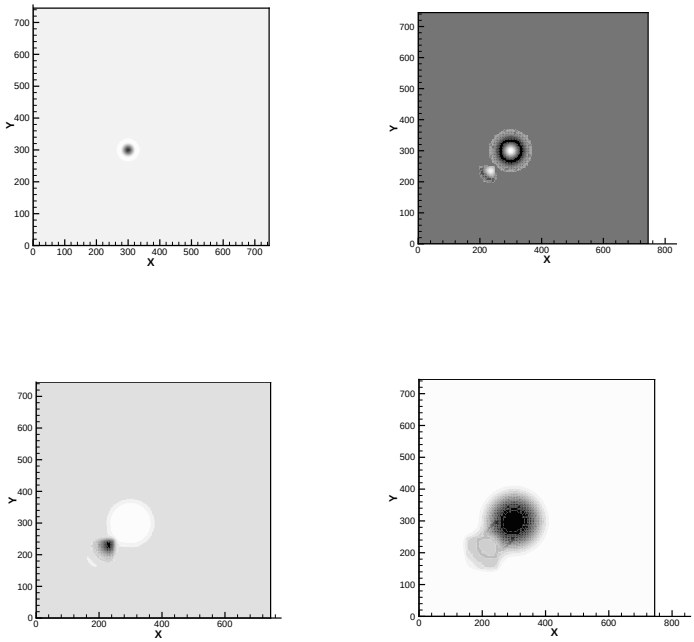

FIG. 15. *Plane slices in xy for depth of* $z = 250$ *m for the blocky inclusion problem run on a* $5 \times 5 \times 5$ *processor grid. Figures correspond to pressure snapshots at times* 0, 0.025, 0.05, 0.075 *seconds. This problem had a blocky high-velocity cube embedded in a homogeneous background model.*

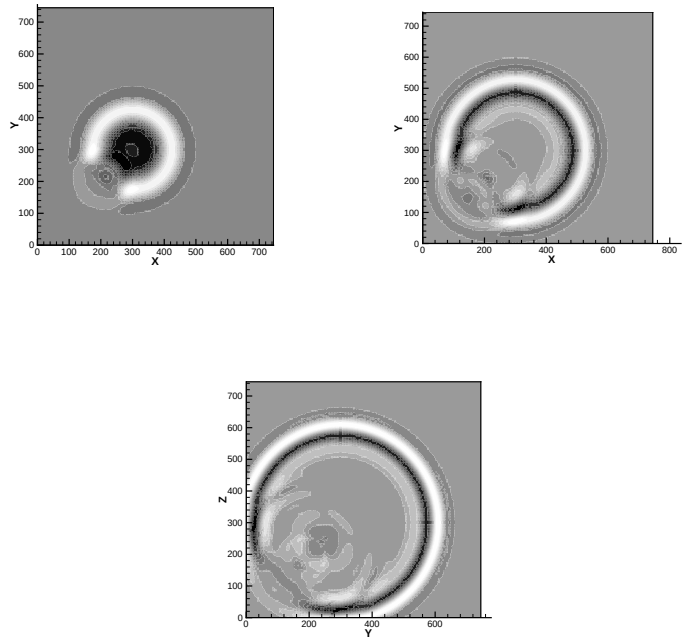

FIG. 16. *Same plane slice in xy as in Figure* 15. *However, snapshots correspond to later times of wave propagation (namely,* 0.1, 0.125, 0.15 *seconds). This problem had a blocky high-velocity cube embedded in a homogeneous background model.*

the computation are to use staggered finite differencing (to obtain additional accuracy without additional storage) and spatial parallelism. In this paper the set of nine isotopic elastic velocity-stress wave equations are discretized using staggered finite differences which are second-order accurate in time and fourth-order accurate in space. The simulation code models multiple source/receiver types. We decompose the physical domain via spatial (or data) parallelism allowing for 1D, 2D, and 3D processor decompositions of any configuration. Each processor allocates memory for its subdomain and four plane faces of padding (ghost cells) per cube face to allow passage of information to neighbors for independent finite difference calculations. The parallel implementation uses the MPI library for portability across platforms with i/o handled via MPI *broadcast* and *reduce* and the finite difference algorithm using simple *sends/receives* for communication. We present three sets of timing studies. The first shows a fixed-sized problem of size $48 \times 48 \times 48$ run on varying numbers of processors of a (mp) Intel Paragon machine. The second study is of scaled-speedup for a problem which varies in size depending upon the number of processors used. Performance on the mp machine is outstanding—with scaled speedup matching the ideal speedup curve even out to 128 processors. For comparison, we ran a scaled problem (size $50 \times 50 \times 50$ on each node) on a Linux cluster. The performance degrades on the cluster due to higher bandwidth and latency costs relative to the mp machine. The serial code is flexible in that the user need not place sources, slice output, etc. at grid point locations. However, this flexibility requires extra communication which seriously impacts efficiency on the cluster. The parallel implementation is accurate to all decimal digits of precision as illustrated by two numerical experiments—a layered earth example and a homogeneous medium problem with a high-velocity blocky inclusion.

Acknowledgments. I thank David Aldridge in the Geophysics Dept. at Sandia National Labs, who wrote the serial code and advised me on its use. I would also like to thank both David Aldridge and Marianne Walck of the Geophysics Dept. for funding this work. Neill Symons and Rory O'Connor helped me run the code on the Linux cluster and understand its performance on that machine. Thanks to Bill Symes of Rice University for preliminary readings of this paper. Finally, Steve Plimpton of the Parallel Computer Science Dept. at Sandia National Labs provided me with some very helpful simplifying suggestions for the parallel implementation.

REFERENCES

[1] K. Aki and P. Richards, *Quantitative Seismology, Theory and Methods*, W. H. Freeman, San Francisco, CA, 1980.

[2] G. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, in Conference Proceedings, AFIPS, 1967, pp. 483–485.

[3] A. Bayliss, K. Jordan, B. LeMesurier, and E. Turkel, *A fourth-order accurate finite-difference scheme for the computation of elastic waves*, Bull. Seismol. Soc. Amer., 76 (1986), pp. 1115–1132.

[4] S. Emerman, W. Schmidt, and R. Stephen, *An implicit finite-difference formulation of the elastic wave equation*, Geophysics, 47 (1982), pp. 1521–1526.

[5] R. Graves, *Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences*, Bull. Seismol. Soc. Amer., 89 (1996), pp. 1091–1106.

[6] S. Hestholm, *3D finite-difference viscoelastic wave modeling including surface topography*, Geophys. J. Internat., 139 (1999), pp. 852–878.

[7] S. Hestholm and B. Ruud, *3D finite-difference elastic wave modeling including surface topography*, Geophysics, 63 (1998), pp. 613–622.

[8] A. Levander, *Fourth-order finite-difference P-SV seismograms*, Geophysics, 53 (1988), pp. 1425–1436.
[9] Y. Luo and G. Schuster, *Parsimonious staggered grid finite-differencing of the wave equation*, Geophys. Res. Lett., 17 (1990), pp. 155–158.
[10] R. Madariaga, *Dynamics of an expanding circular fault*, Bull. Seismol. Soc. Amer., 66 (1976), pp. 639–666.
[11] P. Moczo, M. Lucka, J. Kristek, and M. Kristekova, 3*D displacement finite differences and a combined memory optimization*, Bull. Seismol. Soc. Amer., 89 (1999), pp. 69–79.
[12] K. Olsen, R. Archuleta, and J. Matarese, *Three-dimensional simulation of a magnitude* 7.75 *earthquake on the San Andreas Fault*, Science, 270 (1995), pp. 1628–1632.
[13] P. Pacheco, *Parallel Programming with MPI*, Morgan-Kaufmann, San Francisco, CA, 1997.
[14] A. Pitarka, 3*D elastic finite-difference modeling of seismic motion using staggered grids with nonuniform spacing*, Bull. Seismol. Soc. Amer., 89 (1999), pp. 54–68.
[15] A. Sei, *A family of numerical schemes for the computation of elastic waves*, SIAM J. Sci. Comput., 16 (1995), pp. 898–916.
[16] G. Sleefe, D. Aldridge, G. Elbring, J. Claassen, H. Garbin, and R. Shear, *Advanced Subterranean Warfare (ASW) for Deep Underground Structures*, Technical Report SAND98-0988, Sandia National Labs, Albuquerque, NM, 1998.
[17] J. Virieux, *SH wave propagation in heterogeneous media: Velocity-stress finite difference method*, Geophysics, 49 (1984), pp. 1933–1957.
[18] J. Virieux, *P-SV wave propagation in heterogeneous media: Velocity-stress finite difference method*, Geophysics, 51 (1986), pp. 889–901.