Distributed 3D Finite Difference Modeling of Wave Propagation in Acoustic Media

Alberto Villarreal

Department of Geophysics and Center for Wave Phenomena

Colorado School of Mines

Golden, Colorado 80401, USA avillarr@mines.edu

John A. Scales

Department of Geophysics and Center for Wave Phenomena

Colorado School of Mines

Golden, Colorado 80401, USA jscales@mines.edu

(October 31, 1996)

Abstract

Finite difference modeling of wave propagation in heterogeneous media is a useful technique in a number of disciplines, including earthquake and oil exploration seismology, laboratory ultrasonics, ocean acoustics, radar imaging, non-destructive evaluation, and others. However, the size of the models that can be treated by finite difference methods in three spatial dimensions has limited their application to supercomputers. We describe a finite difference domain-decomposition method for the 3D acoustic wave equation which is well-suited to distributed parallelization. We have implemented this algorithm using the PVM message passing library, and show here benchmarks on two different distributed memory architectures, the IBM SP2 and a network of low-cost PC's running the Linux operating system. We present performance measurements of

this algorithm on both the low bandwidth PC network (10Mbs ethernet) and the high bandwidth SP2 cluster (40 Mbs switch). These results demonstrate the feasibility of doing distributed finite difference acoustic modeling on networks of workstations, but point to the substantial efficiencies that can be expected as higher bandwidth networks become available.

INTRODUCTION

Finite difference and finite element methods are the most direct approaches to performing realistic wave propagation modeling in complex media, providing a complete description of wavefields (including head waves, diffractions and surface waves) with essentially arbitrary variation of material properties. However, the main limitation in the finite difference technique is the great amount of computation and memory required on grids fine enough to meet the constraints of accuracy, stability, and minimum grid dispersion. In order to get some understanding of the scale of the computational demand, let us consider the requirements for a typical application in oil exploration geophysics: Perform seismic forward acoustic modeling in an earth model with dimensions 3 km $\times$ 3 km $\times$ 3 km, with acoustic velocities varying from 1.5 km/s to 4.5 km/s. If we consider the dominant wavelength to be 60 m (which corresponds to a source with a peak frequency of 25 Hz for the slowest wavespeed in the model), then the problem size in wavelengths is $50 \times 50 \times 50$. Let us assume we perform acoustic forward modeling for 4 s on this model using an explicit finite difference algorithm; then, for 12 grid points/wavelength, we will require a $600 \times 600 \times 600$ grid and 7200 time steps. This makes a total of 1.5×10^{12} grid point evaluations. Considering that the calculation to update each grid point requires about 25 floating point operations, then the complete calculation will take 38×10^{12} floating point operations, which means about 3.8×10^5 seconds (approximately 4 days) for a single source in a multisource simulation running on a 100 Mflop workstation (assuming that approximately 1.2 Gigabytes of RAM is available).

In this work the problem of computational demand is addressed using high order finite difference operators and parallel processing techniques. The traditional approach to parallelizing finite difference algorithms has been to use the SIMD paradigm with a fine grained partition, as for example in Myczkowski *et al.* [1] and Fricke [2].

Our approach is to perform a coarse spatial domain decomposition of the problem, dividing the finite difference grid into groups of adjacent points and to assign each one of these subdomains to a different workstation in a network of workstations. This approach permits

us to adapt the size of the groups of points to the available hardware and/or other constraints of the model itself (such as boundary conditions, inhomogeneities, and nonuniform grids).

Considering the ubiquity and low cost of networked workstations, we want to explore the feasibility of performing 3D modeling in such an environment. Our current workstation cluster consists of a number of Linux-based PCs (The use of Linux-based PCs as scientific workstations was described in two articles in the January/February 1996 *Computers in Physics*, [3] and [4]). Our goal is to develop scalable finite difference modeling algorithms on distributed memory heterogeneous networked workstations, as well as to study hardware performance-related issues involved in the execution of these algorithms. Issues such as the computation/communication ratio for different problem sizes and/or different number of processors could permit us to predict the behavior of the algorithms when different hardware is used. This will indicate the feasibility of using low cost hardware as a platform for developing and testing parallel software capable of scaling to faster hardware.

As a starting point, a distributed memory algorithm for 3D acoustic modeling is designed using a message-passing programming model and is implemented using the PVM library as the message passing interface [5]. We show benchmarks of this algorithm on two different parallel systems. These results will be used to design a distributed elastic modeling algorithm, which is even more challenging in terms of computation and communication requirements.

A portion of this paper is devoted to a brief introduction to parallel architectures and parallel programming models. Following this overview, details of the finite difference acoustic modeling are discussed. Next, a distributed implementation of this algorithm is presented, followed by benchmark results showing the parallel efficiency of the algorithm compared to single processor executions.

PARALLEL ARCHITECTURES AND PROGRAMMING MODELS

Parallel Architectures

Multiprocessors, or *shared memory MIMD* (Multiple Instruction Multiple Data) computers, are perhaps the best known parallel computer architectures. The term MIMD means that each processor can execute a separate stream of instructions on its own data, and shared memory implies that all processors share access to a common (usually large) memory. Examples of this class of machine include the SGI Power Challenge, the Cray Y-MP and C-90, and many multiprocessor workstations.

Distributed memory MIMD computers comprise another important class of parallel computer systems. Distributed memory means that memory is distributed among the processors, rather than placed in a central location, as in multiprocessors. Here, each processing node executes its own program. This program may access local memory and may send and receive messages to or from other nodes. Messages are used to communicate with other nodes, which means to read and/or write remote memories. The CM-5 and the Cray 3TD are examples of this type of MIMD machine.

A more specialized class of distributed memory architecture is the SIMD (Single Instruction Multiple Data). In these machines, all processors execute the same instruction stream on different pieces of data. This architecture is best suited to problems characterized by a high degree of regularity. The CM-2 is an example of this architecture, although MIMD machines such as the CM-5 can also emulate the SIMD characteristics.

Another class of MIMD parallel computer system is a cluster of networked workstations (NW). It can be built as a group of dedicated processors linked by a fast network or switch (as in the IBM SP2 cluster), or it can consist of a dynamically varying set of networked stand-alone workstations that perform long running computations during idle periods.

Modern processors commonly used in workstations today incorporate supercomputer technology to a certain degree, such as pipelining and other techniques, and in theory, aggreg-

ated memory and computing power in a cluster could approach supercomputer performance. Because NW's are created from standard systems, they are easy to scale. For example, as fast workstations become available they can be added to the system just by plugging them into the network. However, a critical issue in these systems is the network. Applications with high synchronization and communication requirements are likely to run inefficiently in an NW if the network is slow.

There have been a number of parallel implementations of wave equation modeling algorithms. Most of them have used the shared memory or SIMD architectures. Vafidis *et al.* [6] have presented results of elastic 2D finite difference modeling in transversely isotropic media using a fully vectorized algorithm. Vector computation can generate important speed-ups in finite difference algorithms, although until recently this capability was present only in costly supercomputers. Other algorithms targeted to SIMD distributed memory architectures have been published, as for example in Myczkowski *et al.* [1], who describes an efficient 2D acoustic modeling code for a CM-2 system.

There have been several distributed implementations of geophysical algorithms in NW's. For example Black *et al.* [7] showed a study of a 3-D depth migration on a network of RISC workstations. Almasi *et al.* [8] showed performance results of a 2D inverse scattering via a depth extrapolators algorithm implemented in a NW. Bunge and Baumgardner [9] have performed thermal convection calculations for the Earth's mantle using a message-passing algorithm on a cluster of workstations. Ewing *et al.* [10] showed results of a PVM implementation of 2D acoustic and elastic Finite-Difference seismic modeling in a homogeneous dedicated cluster of IBM RS/6000 computers, and Olsen *et al.* [11] conducted elastic earthquake simulations via domain decomposition on a massively parallel computer.

Let us mention some of the factors which led us to consider NW's as our computer platform:

- Low cost and availability.

  Given the low cost per node of Unix workstations, these networks are becoming uni-

versal. And often when these workstations are used for interactive work, they remain idle during nights and weekends and thus become available for performing long-running computations during those idle periods with no extra cost.

- Software portability.

  Given the high degree of standarization present in Unix workstations, it is possible to design portable software able to execute in a broad range of architectures. This means that the distributed application will take advantage of a heterogeneous network of workstations, rather than being constrained by a single architecture. Another important issue regarding portability is related to the portability of the message passing software itself. There are several freely available portable message-passing libraries. A popular example is Oak Ridge National Laboratory's PVM (Parallel Virtual Machine) [5], which we have selected to build our parallel system.

- Software scalability.

  Parallel software must be scalable (i.e. the size of the problem we are trying to solve, or the performance of the application, must increase when more resources become available) in order for the application to be cost-effective. And given that NW's are scalable, either by increasing the number of nodes or the bandwidth of the network, then it is possible to use a NW to design and test parallel systems with the idea of porting them to a larger NW or to a Massively Parallel system.

We must make clear, though, that by exploring the feasibility of solving a wave propagation problem in a NW, we do not mean to replace high performance computational resources such as supercomputers, which because of lower latencies and higher bandwidths are more efficient for applications with large communication requirements. The NW can provide a low cost solution to many problems, and it can also improve the effectiveness of supercomputer resources by permitting them to be used for the most demanding applications while the NW can be used as a development and testing platform for parallel applications. However,

advances in networking technology and processor performance are expanding the set of applications that can be executed efficiently on NW's.

Programming Models

A parallel programming model is a means to describe parallel algorithms for a certain computer architecture. For example, in the shared memory programming model, it is assumed that the data are in a memory which can be accessed by any processing element, and so it is well suited for shared memory machines. In the message passing programming model (which is a special case of distributed memory programming models), a parallel computation consists of several tasks executing concurrently, and every task can read and write its own memory, as well as remote memories (located in other processing nodes) by sending/receiving messages (a message is a set of bytes transferred from one computer to another computer), thus being appropriate for distributed memory architectures. However, with the advent of new programming tools such as message passing libraries and distributed shared memory systems, algorithms designed in a certain programming model are not restricted to execution in a particular architecture.

In this work, the finite difference algorithm will be designed using the message passing programming model. Implementing this algorithm making use of a portable message passing library, such as PVM, will permit us to execute the code in any machine in which PVM is available, including shared memory multiprocessors and Massively Parallel Computers (MPP).

PROBLEM STATEMENT: 3D FINITE DIFFERENCE WAVE PROPAGATION MODELING

A basic problem in seismology and other fields is the determination of the motion on the free surface and in the interior of a layered acoustic or elastic medium caused by an

impulsive source. In what follows we will assume that the medium is acoustic and its density is constant.

Modeling the motion of the medium

In that case the equations of motion for the medium are

$$\frac{\partial^2 p}{\partial t^2} + s(x_i) = c^2 \frac{\partial^2 p}{\partial^2 x_i}. \tag{1}$$

where p is the pressure, x_i, $(i = 1, 2, 3)$ are the three cartesian coordinates (later labeled x, y, and z), t is the time, and $s\,(x_i)$ is the source term. The wave can be initiated using initial conditions, defined by p and $\partial p/\partial t$ at $t = 0$, or using an explicit source function $s(x_i)$. In this case an explicit source function will be used.

The boundary conditions for this equation can be of several types (absorbing, free surface, periodic). Absorbing boundary conditions are used here.

Discretization of the model

We divide the model into a grid of N_x by N_y by N_z points. If we call Δx, Δy and Δz the distance between points in the grid in the x, y and z axes respectively, then we get $x = n_x \Delta x$, $y = n_y \Delta y$ and $z = n_z \Delta z$, where $n_x = 1, 2, ..., N_x$, $n_y = 1, 2, ..., N_y$ and $n_z = 1, 2, ..., N_z$. Also if Δt is an increment in time, then $t = k\Delta t$ where k is the time step with $k = 1, 2, ...$.

Given this discretization, we can now describe a finite difference scheme to approximate the solution of the acoustic wave equation.

Numerical solution of the wave equation using finite differences

We substitute an qth-order central difference operator for the second space derivatives in the wave equation, and a second-order central difference operator for the second time derivatives, to obtain the discretized finite difference equation. A general framework for

deriving higher-order finite difference schemes was proposed by Dablain [12]. He expresses the qth-order centered finite difference operator for the second space derivative with respect to x_i as a weighted sum:

$$\frac{\partial^2 p}{\partial x_i^2} \simeq \frac{1}{\Delta x_i^2}[w_0 P_r + \sum_{k=1}^{q/2} w_k (P_{r+k} + P_{r-k})] \tag{2}$$

where P_r are the values of the pressure p at the discrete position r, in the ith direction. The weights w_k are derived from the power series expansion of P_r and further simplification. The details of this calculation and values of coefficients w_k will not be given here.

For the second time derivatives, we have the second-order central difference operator:

$$\frac{\partial^2 p}{\partial t^2} \simeq \frac{1}{\Delta t^2}[P_r^{k+1} - 2P_r^k + P_r^{k-1}]. \tag{3}$$

Substituting these operators into the wave equation, we obtain an explicit formula for the values of the wavefield at time $k+1$. For example, for $q = 2$ we obtain the second-order (in space) scheme:

$$\begin{aligned} P_{l,n,m}^{k+1} = r_{l,m,n} \, (& P_{l-1,n,m}^k + P_{l+1,n,m}^k + \\ & + P_{l,n-1,m}^k + P_{l,n+1,m}^k + \\ & + P_{l,n,m-1}^k + P_{l,n,m+1}^k - 6 \, P_{l,n,m}^k) + \\ & + 2 \, P_{l,n,m}^k - P_{l,n,m}^{k-1} + S_{l,n,m} \end{aligned} \tag{4}$$

where $P_{l,n,m}^k$ is the value of the pressure at time $k\Delta t$ at position $(n_x \Delta x, n_y \Delta y, n_z \Delta z)$,

$$r_{l,n,m} = c_{l,n,m}^2 \left(\frac{\Delta t^2}{\Delta x^2} \right), \tag{5}$$

$c_{l,n,m}$ is the velocity at $(n_x \Delta x, n_y \Delta y, n_z \Delta z)$ and $S_{l,n,m}$ is the discrete source term.
Similarly, for $q = 4$ we obtain the fourth-order (in space) scheme:

$$\begin{aligned} P_{l,n,m}^{k+1} = \hat{r}_{l,m,n} \, [& P_{l-2,n,m}^k + P_{l+2,n,m}^k + P_{l,n-2,m}^k + \\ & + P_{l,n+2,m}^k + P_{l,n,m-2}^k + P_{l,n,m+2}^k - \\ & - 16 \, (P_{l-1,n,m}^k + P_{l+1,n,m}^k + P_{l,n-1,m}^k + \end{aligned}$$

$$+ P^k_{l,n+1,m} + P^k_{l,n,m-1} + P^k_{l,n,m+1}) +$$
$$+ 90\, P^k_{l,n,m}] +$$
$$+ 2\, P^k_{l,n,m} - P^{k-1}_{l,n,m} + S_{l,n,m} \tag{6}$$

where

$$\hat{r}_{l,n,m} = \frac{-\left(c^2_{l,n,m} \frac{\Delta t^2}{\Delta x^2}\right)}{12} \tag{7}$$

and where, in both examples, we have assumed that $\Delta x = \Delta y = \Delta z$, although the method is not restricted to this case.

Absorbing boundary conditions based on gradual reduction of the amplitudes in a strip of nodes along the boundaries are used in this implementation [13].

Constraints on the discretization of the model

There are several issues involved in the choice of the grid parameters in the discretization step. The first problem arises from the fact that in order to keep the model numerically stable, the ratio $(\Delta t/h)$, where h is the maximum grid spacing, must be small. Specifically, the largest time step which must be used to guarantee stability is [14]

$$\max(\Delta t) = \frac{\mu \max(\Delta x, \Delta y, \Delta z)}{\max(c_i)}$$

where c_i, $(i = 1, 2, ...)$ are the acoustic wavespeeds involved in the computation, and μ is a constant that depends on the order of the method used.

The second problem, grid dispersion, comes from the fact that wave propagation in discrete systems is inherently dispersive. The way to minimize grid dispersion is to make the wavelengths long compared to the grid spacing. It has been observed empirically that using at least 10 grid points per the minimum wavelength involved in the computation for a second-order scheme, or at least 4 grid points per minimum wavelength for a fourth-order scheme will keep the grid dispersion at values sufficiently low so that it does not disturb the modeled wavefield in typical geophysical applications. More detailed quantitative analysis

of the dispersion problem, including the relation between operator length and the required number of grid points per shortest wavelength for a required accuracy, is given by Holberg [15] and Marfurt [16].

Finally, the accuracy of this numerical method, whose sources of error come from the replacement of continuum derivatives by finite difference operators, directly depends on the spatial and temporal sampling. Given that these values must be small in order to meet the criteria for stability and grid dispersion discussed above, then in typical geophysical applications the accuracy of the results will be acceptable.

METHOD

Sequential algorithm

The conventional sequential algorithm for extrapolating the wavefield looks as follows:

```
BEGIN
   - set up  data structures,
     variables and constants.
   - read velocity and density models.
   FOR every time step DO
       FOR every grid point DO
           - Update the wavefield.
       END
   END
END.
```

Distributed algorithm design

Domain Decomposition

Domain decomposition involves assigning subdomains of a full computational grid to separate processors and solving the equations for each subdomain concurrently. Grids used to represent models in finite difference applications are natural candidates for simple domain decomposition techniques because of their regularity. This is especially evident in an explicit finite difference scheme, which can be finely partitioned to expose the maximum possible concurrency (i.e. defining a task for each grid point), because there are no data dependencies when computing the wavefield at different grid points for a given time step.

For our purposes, the grid can be partitioned in a coarser way, agglomerating a whole set of grid points (a subdomain) in a single task. Several possibilities exist for dividing the model into subdomains. Figure (1) shows two possible decompositions for a three-dimensional grid. The most natural decomposition for one-dimensional communication hardware (such as the Ethernet) is a one-dimensional decomposition in the direction with the largest number of grid points, as shown in Figure (2), assigning each subdomain to a different processor. A one-dimensional decomposition is used in this work for the present algorithm, and it will be tested in models with different geometry to investigate the effects of these differences on the performance. In the future we plan to test higher order decompositions.

[FIG. 1 about here.]

[FIG. 2 about here.]

[FIG. 3 about here.]

Interprocessor communication

In the coarse-grained partition shown in Figure (2), each individual task (running in an individual processor) approximates the wavefield at each grid point in the subdomain at time

$k+1$, based on the values of the wavefield at time steps k and $k-1$ at the same gridpoint and its adjacent neighbors. We immediately see that the wavefield can be updated simultaneously in all of the subdomains, except in those gridpoints at the borders since their neighbors are in a different layer. In this case the value of the wavefield at those neighboring gridpoints must be replicated in the contiguous layer, Figure (3), in order for the wavefield in that contiguous layer to be totally defined, and so to guarantee the continuity of the wavefield across the subdomains. This is the essence of the parallel algorithm. At each time step, the border layer of each subdomain must be interchanged between every two adjacent subdomains. And the thickness of this layer will depend on the order of the finite difference operator; e.g., the thickness will be 1 grid point for a second-order operator, 2 grid points for a fourth-order operator, etc.

A basic algorithm

In summary, the parallel algorithm using the domain decomposition described can be represented as follows:

```
BEGIN
- set up  data structures, variables
  and constants.
- read velocity and density models.
- set up a partition of the grid
  according to the number of available
  CPU´s.
- Create a task for each element
  (subdomain) of the partition and
  send to each task its correspondent
  density and velocity values.
```

```
  FORALL tasks simultaneously DO
    FOR every time step DO
       FOR every grid point DO
            - Compute wavefield
       END
       - Interchange border grid points
         with each adjacent layer.
    END
  END

  - gather wavefield information from
    every task.
END.
```

This algorithm presents particular problems for parallel execution. It is a memory and communication intensive problem, especially if low-speed networks are to be used. An important amount of interprocessor communication is generated at every time step, considering that data must be interchanged between neighboring subdomains at every time step in order to maintain continuity of the extrapolated wavefield.

Although there is no explicit synchronization in the algorithm, it is clear that every processor will have to stop while it waits for the border grid points from the adjacent layers, thus reducing the overall efiiciency of the parallel computation. This waiting time will increase with the number of processors. In the next section we discuss a way to reduce this effect.

An improved algorithm

In order to minimize the impact of interprocessor communication on the performance of the algorithm, we must always look for opportunities for overlapping computation and communication. In this case, we can observe in the pseudocode that the data contained in the messages that are being interchanged between these tasks will be used only in the computation of the borders of each subdomain, and the computation of the inner grid points can take place while the messages are traveling across the network. Every task can update the border grid points, send these values to the other tasks, and start computing the inner grid points without waiting for the new data to arrive. After computing the inner grid points, the task must wait for the other tasks' information before updating again its border grid points. Depending on the speed of the network and processors, the new values could already be available in local memory by the time the task is ready to access them, thus greatly reducing the waiting time for each task. Notice here that the bigger the subdomain (i.e. the larger the number of grid points each task has to update, and so the longer that task is busy updating those grid points), the better the opportunities for a total overlapping between computation and communication.

Taking this overlapping procedure into account, the FORALL loop of an improved algorithm will look like this:

```
FORALL tasks simultaneously DO
  FOR every time step DO
    FOR every border grid point DO
      - Compute wavefield
    END
    - Send values of updated border
      grid points to each adjacent
      layer.
```

```
    FOR every inner grid point DO
      - Compute wavefield
    END
      - Receive values of updated
        border grid points from each
        adjacent layer.
  END
END
```

Again, as in the first algorithm, there is no explicit synchronization between the tasks, although the fact that each processor must wait for new data before computing a new time step at the border grid points, induces some degree of synchronization. A consequence of this is that load imbalances are generated if one of the processors is slower than the others, or if a different task is also using that processor. The current version of this algorithm has been implemented with no load balancing techniques, being more apropriate for overnight or dedicated-time executions.

IMPLEMENTATION

Hardware

The improved parallel algorithm described above was coded and tested on the following hardware:

- A cluster of Pentium-based workstations connected via 10 Mbs Ethernet (with a ring topology). Each workstation has 32 Mbytes of RAM and runs under the Linux operating system [17]. The clock speed of the Pentium processors ranges between 90 and 120 Mhz.

- An 8-node IBM SP2 connected via a 40 Mbs high performance switch. Each node has 256 Mbytes of RAM and the clock speed of each processor is 66.7 Mhz.

In the case of the Pentium cluster, both the network and the machines were not run in a dedicated mode for these tests, and so the results presented here are representative of a realistic computational environment. However, the tests were run overnight in order to minimize the impact on other users. In the case of the IBM SP2, dedicated processors and network were used for the simulations.

Software

As mentioned before, the algorithm was designed using the message passing paradigm. The specific strategy for the workload allocation we used was the so-called *node only* mode, in which multiple instances of the program (which resides in the common file system of the clusters) execute and each process generated in this way will update one of the subdomains of the partitioned grid, as well as establishing the communication with its two neighbors. In this strategy, one of the processes (the one initiated manually by the user) spawns the rest of the processes, and takes over the initial distribution of data, problem parameters and workload allocation, and the final allocation and output of the results, in addition to contributing to the computation itself.

The program was coded in C (single precision), making use of the SU (Seismic Unix) library, and it is partially based on existing CWP (Center for Wave Phenomena, Colorado School of Mines) software, which is freely available from the CWP World Wide Web site [18]. The interprocessor communication, process creation, and synchronization were implemented using the PVM library.

Optimization issues

In order to obtain general performance results, the algorithm was implemented at this time without the benefits of hand tuning. Although this decreases the performance of the algorithm, it permits us to compare performance results obtained in several architectures with more emphasis on the efficiency of the communication pattern designed for the general problem of distributed implementation of finite differences. Specific hardware-dependent (pipelining, vectorization, etc.) or model parameter-dependent (moving grid) optimizations must be applied to this software when used in a production environment.

Regarding the performance of the message passing library, we used the freely available version of PVM for the same reason mentioned above. However, several optimized architecture-dependent implementations of PVM exist, and they should be used for production executions. A study of comparative performance of PVM versions on different hardware is given in Casanova *et al.* [19]. Also the PVM group library was used to implement synchronization and communication aspects of the algorithm. This library provides facilities for handling groups of tasks and facilitates coding and debugging, although it also decreases the performance of the algorithm, especially when a slow network and a large number of machines are used. No load balancing was attempted at this point, mainly because the Pentium-based cluster has only a small degree of heterogeneity, while the IBM SP-2 cluster is totally homogeneous.

EXAMPLE OF A MODELING EXPERIMENT

Figure (4) shows a simplified 3D earth model in the form of a cube with an interface in the middle separating a high velocity layer above ($v = 3000$ m/s) and a low velocity layer below ($v = 1500$ m/s). The model's dimensions are 1 Km each side, and it has been discretized using a 3D grid containing $200 \times 200 \times 200$ grid points. An explosive source was set off at the high velocity layer at short distance from the interface. The simulation was run in our Linux network. Figure (5) shows different views of a 3D snapshot. The generation of

the so-called *non-geometrical waves* (pseudospherical and leaking waves) in the low velocity medium becomes evident in these pictures.

[FIG. 4 about here.]

[FIG. 5 about here.]

PERFORMANCE RESULTS

Parallelization experiments using a one-dimensional decomposition were done to examine scaled speedup of the parallel algorithm, i.e. the speedup obtained using a scaled-size problem. In a scaled-size problem, the size of the problem (the total number of grid points in our case) increases as processors are added, thus keeping the amount of work done by each processor independent of the number of processors (Figure 4). In contrast, in fixed-size problems, as the number of processors increases the amount of work done by each processor decreases, making more difficult the task of evaluating the effects of the communication in the performance of the algorithm.

Our one-dimensional decomposition experiments were done with decomposition in the z direction, although the same decomposition can be applied in the x or y directions, depending on the direction of the longest axis and/or specific constraints of the model. Subdomain sizes were fixed by the memory available in each processor. For the Pentium cluster every subdomain contained approximately 800,000 grid points, which in the sequential case (when running on one CPU) corresponds to a cubic grid with sides $93 \times 93 \times 93$, thus permitting one to scale the size of the complete grid to a 200 grid point cube when 10 processors are being used, as shown in Figure (6). Also a brick (parallelepiped) geometry was used, with the same 800,000 grid points in each subdomain, but distributed in such a way that the side of the brick where the decomposition is taking place (i.e. the z direction) is twice as large as the other two sides (Figure (6)). For the IBM SP2 cluster, every subdomain contained approximately 4×10^6 grid points (a cubic grid with sides $159 \times 159 \times 159$ in the sequential

case). Second-order and fourth-order finite difference schemes with cubic and brick model geometries each were used for simulations in each architecture. One to 8 processors were used for the experiments in both the Pentium network and the IBM SP2.

[FIG. 6 about here.]

The wave propagation simulation parameters were held constant for all the experiments; in particular, holding the grid point spacing constant (for a scheme with a given order) allowed the physical dimension to increase in a given direction when the number of grid points increased in that direction. A homogeneous medium with acoustic wavespeed of 3,000 m/s was used for the simulations, and the wavefield propagation was simulated for 0.5 sec, with a time step of 0.67 msec (the computation time in this finite difference simulation is independent of the degree of heterogeneity of the medium, so our timing results will be equally applicable to models with arbitrary heterogeneity). A source wavelet with a maximum frequency of 75 Hz (which corresponds to a minimum wavelength of 40 m) was used to represent the source. The spatial grid point spacing for the second-order scheme was 4 m (in the three directions x, y and z), and 10 m for the fourth-order scheme. These parameters imply that the real size of the model we are working with when using 8 PC nodes is a 730 m cube when we use the second-order scheme, and a 1,800 m cube when we use the fourth-order scheme. These parameters satisfy the Courant-Friedrichs-Levy condition. Following are the speedup measurements we obtained by running the codes in several architectures.

The Pentium network

Table (I) shows the scaled sizes for the models used in the experiments corresponding to a cubic geometry. Figures (7) and (8) show the results obtained for the simulations performed using the number of processors and sizes shown in Table (I), with a second-order and a fourth-order scheme, respectively.

[TABLE 1 about here.]

[FIG. 7 about here.]

[FIG. 8 about here.]

In these plots, the line labeled *Sequential* indicates the elapsed time for the sequential execution of the basic model (first entry in Table (I)). The curve labeled *total* indicates the total elapsed time for the execution of the parallel algorithm (only the time integration loop was considered for the measurements, because that is the only part which executes in parallel). The numbers above each point in the *total* curve indicate the efficiency of the parallel algorithm obtained using the number of nodes specified in the horizontal axis. This efficiency was computed as follows:

$$efficiency = \left(\frac{parallel\ speedup}{number\ of\ nodes} \right) \times 100\%$$

with

$$parallel\ speedup = \frac{sequential\ time}{parallel\ time}$$

where *sequential time* and *parallel time* correspond to the curves labeled *Sequential* and *Total*, respectively, in the figures. The sequential time shown in the figures was obtained using a 90 Mhz Pentium.

Notice that, as the size of the problem scales with the number of processors (keeping the computational load in every processor constant), the grid running in 8 CPU's has eight times more grid points than the grid running in one CPU, although the elapsed time for this parallel simulation should be the same as the sequential time (the dashed line labeled *Sequential* in the figures) if the computation and communication overheads were negligible. This is not the case here, and we observe that the efficiency of the algorithm is decreased because of the time spent in the extra computation and communication required by the distributed algorithm.

[TABLE 2 about here.]

Table (II) shows the scaled sizes of the models with a brick geometry, and Figures (9) and (10), whose description is the same as the figures previously shown, reflect the results obtained for models with this kind of geometry.

[FIG. 9 about here.]

[FIG. 10 about here.]

The IBM SP-2

As the SP-2 has more memory than the PC's used in the experiments shown in last section, a model with larger dimensions was used. The size of the model was scaled so that the percentage of the total available memory used in these simulations was the same as the one used in the Pentium network.

Table (III) and Table (IV) show the scaled sizes for the models used in the experiments corresponding to a cubic and a brick geometry, respectively. Figures (11) and (12) show the results obtained for the simulations performed using the number of processors and sizes shown in Table (III), with a second-order and fourth-order scheme, respectively, while Figures (13) and (14) show the corresponding results using the data in Table (IV).

[TABLE 3 about here.]

[TABLE 4 about here.]

[FIG. 11 about here.]

[FIG. 12 about here.]

[FIG. 13 about here.]

[FIG. 14 about here.]

INTERPRETATION OF RESULTS

Here are some of the interpretations of these results:

- **Parallel efficiency.** Interprocessor communication overhead and redundant computations are the main limitations for obtaining high performances on workstations. The overhead increases with the number of processors used. Relatively high efficiencies are obtained in workstations when using 5 or fewer nodes, or when using high bandwidth networks.

- **Optimization.** As we mentioned before, the algorithm was implemented avoiding hardware-dependent and model-dependent optimizations in order to get a better basis of comparison between the experiments in different hardware platforms. Thus the benchmarks presented should be regarded as lower bounds on efficiency. We have observed that optimization techniques available in high-end workstations, such as pipelining and cache-oriented programming, can greatly improve the speed of the code. The communication overhead can also be reduced using optimized versions of the message passing libraries, such as the enhanced PVMe library [20] for IBM, or forthcoming freely available versions of PVM, which will include improved communication capabilities.

- **Influence of model geometry and the order of scheme on the efficiency.** Model geometries play an important role in the performance of this algorithm. The cubic geometry seems to generate the lower values of efficiency because the low volume/area ratio of each subdomain generates a low computation/communication ratio. This suggests that a higher order domain decomposition (two or three-dimensional) can be more efficient with this kind of geometry. The order of the finite difference scheme also affects the efficiency because of the increment in communication required by higher order methods. This is evident if we compare the high efficiencies shown in Figure (9), which corresponds to the brick geometry using a second order scheme, with the low efficiencies

shown in Figure (8), corresponding to a cubic geometry using a fourth order scheme. However, using a high order scheme allows the grid to represent models with larger physical dimensions, and so the efficiency of the fourth order scheme in this case can be higher than the efficiency of the second order scheme regarding the physical dimensions of the model.

- **Size of the model.** Considering that computation and communication are partially overlapped in this algorithm, as discussed before, we expect better efficiencies when the size of the subdomains increases. Larger subdomains will increase the amount of computation to update them and also the opportunity for the communication to occur concurrently with the computation. The only limitation for increasing the size of each subdomain is the amount of RAM available in each workstation. As more memory becomes available, larger problems can be handled.

CONCLUSIONS AND FUTURE WORK

We have developed a distributed 3D acoustic finite difference modeling algorithm using a domain decomposition approach, and implemented this algorithm in ANSI C via the freely available PVM message passing library. We have presented benchmarks of this code on two different architectures, a cluster of network connected Linux PC's, and an IBM SP2 cluster. Using a scaled problem (to keep the size of each machine's sub-problem fixed) we have observed parallel efficiencies of from 46-76% on a cluster of 8 workstations (depending on the geometry of the model and the order of the finite difference used) and up to 94% on an 8 processor SP2. With this algorithm we are able to simulate 3D acoustic wave propagation for a model of 6.4×10^6 gridpoints on a network of 8 PC's.

In order to achieve a large measure of portability we have taken advantage of no hardware-dependent optimization. In particular, we believe that considerable efficiencies could be achieved by taking advantage of pipelining and cache-oriented programming. We have limited the size of the subdomains to fit into the available memory of our workstations. As the

resources of our workstation network increase (memory and bandwidth) we will be able to take advantage of these resources by adjusting the size of the subdomains. Up to now we have already developed a modeling code for 2D elastic anisotropic media which uses the same algorithm shown in this work. We are currently benchmarking this new code, which permits us to simulate wave propagation in more realistic elastic, anisotropic media.

ACKNOWLEDGEMENTS

This work was partially supported by the sponsors of the Consortium Project on Seismic Inverse Methods for Complex Structures at the Center for Wave Phenomena, Colorado School of Mines, the Shell Foundation, the Army Research Office under grant DAAH04-95-1-0173, and the National Science Foundation under grant DMS-9506603. We would especially like to thank IBM for access to their SP2 cluster at Poughkeepsie, NY.

REFERENCES

[1] J. Myczkowski, D. McCowan, and I. Mufti, The Leading Edge 49 (1991).

[2] R. Fricke, Geophysics **53**, 1143 (1988).

[3] K. Ballüder, J. Scales, C. Schröter, and M. Smith, Computers in Physics **10**, 17 (1996).

[4] J. Templon, Computers in Physics **10**, 49 (1996).

[5] A. Geist *et al.*, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing* (MIT-Press, ADDRESS, 1994).

[6] A. Vafidis, F. Abramovici, and E. R. Kanasewich, Geophysics **57**, 218 (1992).

[7] J. Black, C. Su, and W. Bauske, in *SEG 91 Expanded Abstracts* (Society of Exploration Geophysicists, Tulsa, Oklahoma 74101, 1991), pp. 353–356.

[8] G. Almasi *et al.*, Concurrency: Practice and Experience **5**, 105 (1993).

[9] H. Bunge and J. Baumgardner, Computers in Physics **9**, 207 (1995).

[10] R. Ewing *et al.*, IEEE Parallel & Distributed Technology **2**, 26 (1994).

[11] K. B. Olsen, R. J. Archuleta, and J. R. Matarese, Science **270**, 1628 (1995).

[12] M. Dablain, Geophysics **58**, 54 (1986).

[13] C. Cerjan, R. Kosloff, and M. Reshef, Geophysics **50**, 705 (1985).

[14] A. Mitchell and D. Griffiths, *The Finite Difference Method in Partial Differential Equations* (John Wiley, ADDRESS, 1980).

[15] O. Holberg, Geophysical Prospecting **35**, 629 (1987).

[16] K. Marfurt, Geophysics **49**, 533 (1984).

[17] At the time these tests were performed we were using Slackware Linux with version 1.3.20 of the kernel.

[18] J. Cohen, The SU User's Manual, 1995, http://www.cwp.mines.edu.

[19] H. Casanova, J. Dongarra, and W. Jiang, Technical Report No. CS-95-301, University of Tennessee (unpublished).

[20] *IBM AIX PVMe User's Guide and Subroutine Reference, Release 3.1*, IBM, Kingston, New York, 1995.

List of Figures

FIGURES

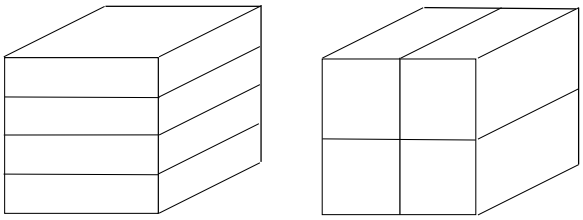

FIG. 1. Two possible domain decompositions for a three-dimensional grid. Each subdomain contains the same number of grid points in both decompositions. A one-dimensional decomposition is used in this work.

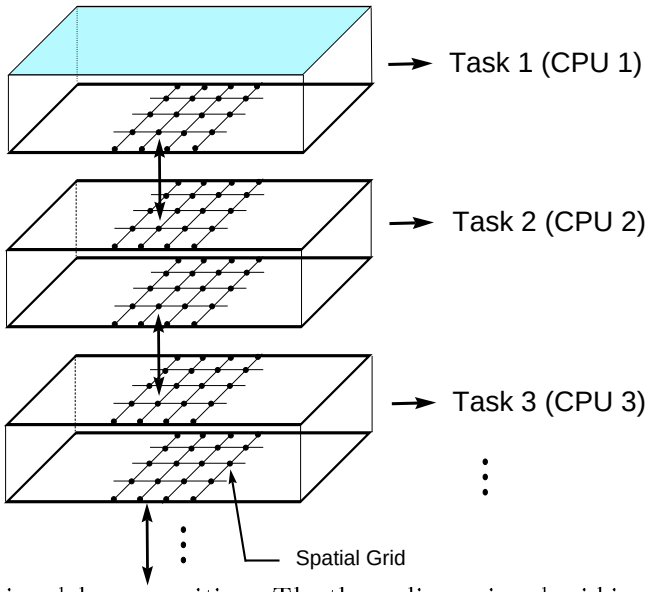

FIG. 2. One-dimensional decomposition. The three-dimensional grid is vertically partitioned into subdomains of the same size. Each subdomain will be updated by a task in a different computational node.

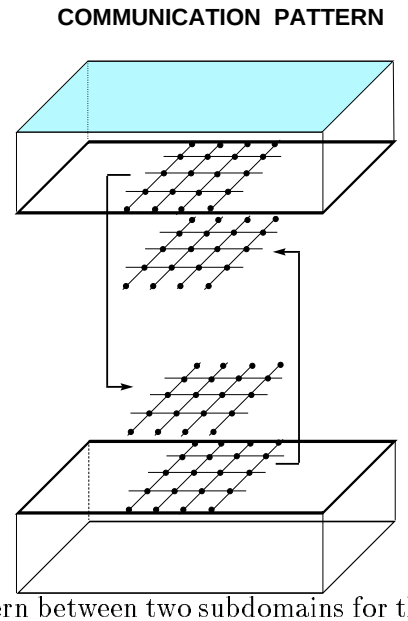

FIG. 3. Communication pattern between two subdomains for the one-dimensional decomposition of our model. The internally computed lower layer of the node above is sent as a message to become the upper boundary of the node below. Likewise, the internally computed top layer of the node below is sent to the node above to become its lower boundary.

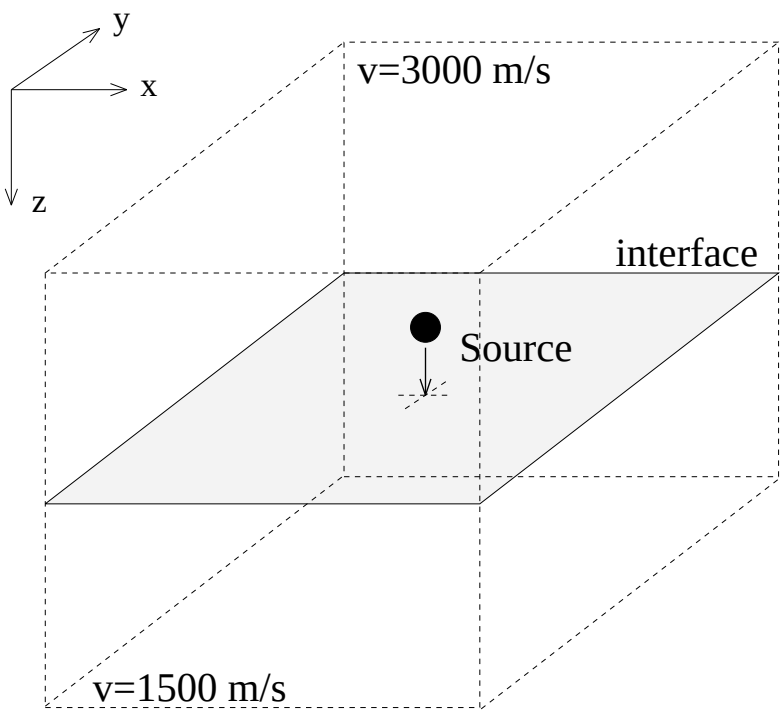

FIG. 4. Simplified earth model with an interface separating a high velocity medium (above) and a low velocity medium (below). The explosive source is set at the high velocity medium close to the interface.

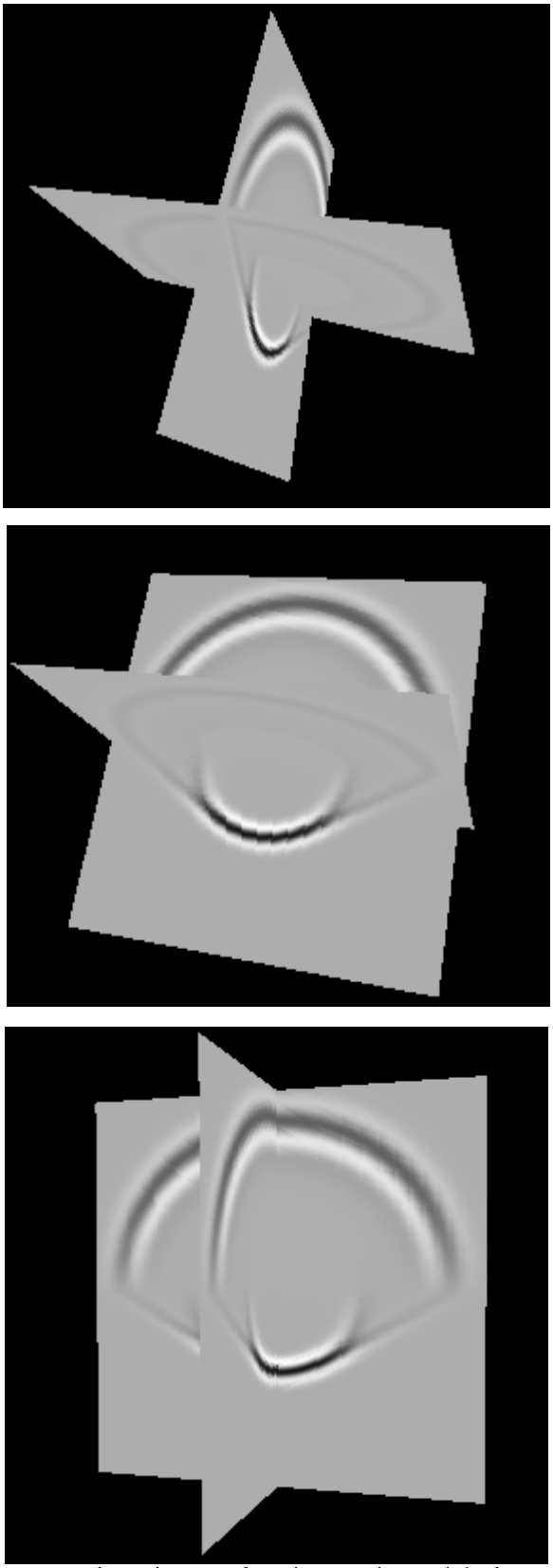

FIG. 5. Snapshot of pressure distribution for the earth model shown in Figure (4). The $x - y$ plane coincides with the interface between the two media.

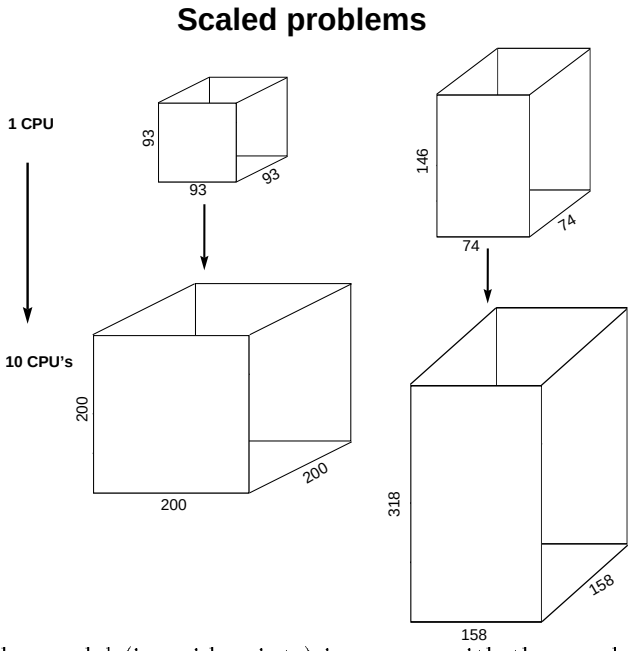

FIG. 6. The size of the model (in grid points) increases with the number of nodes for cubic and brick geometries. In this example the cube and the brick running on 10 CPU's are ten times the size of those running on one CPU in the PC network.

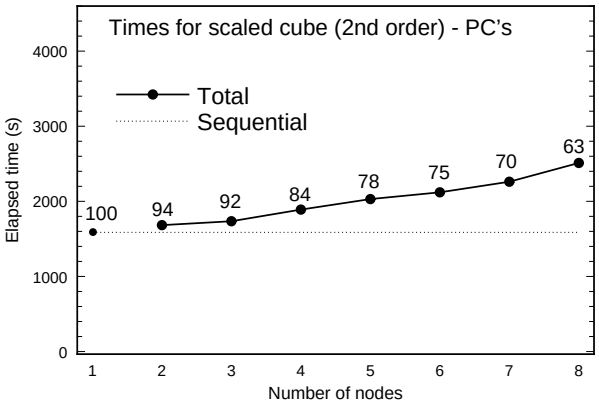

FIG. 7. *Cubic geometry with a second-order scheme.* Measured wallclock times for the experiments using the Pentium network.

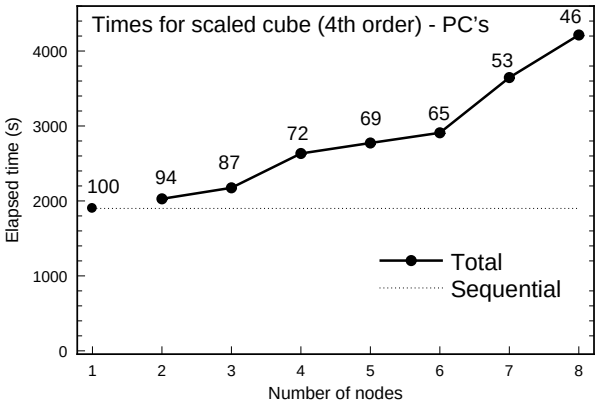

FIG. 8. *Cubic geometry with a fourth-order scheme.* Measured wallclock times for the experiments using the Pentium network.

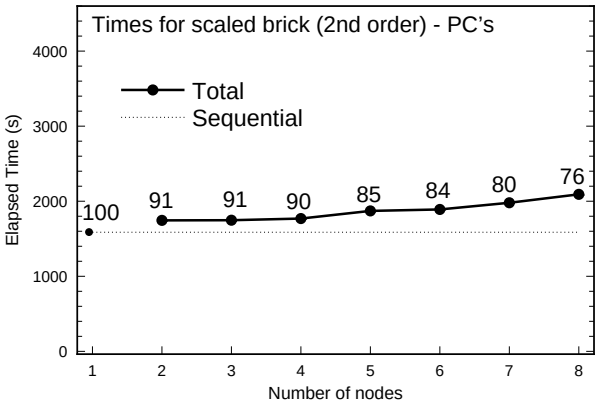

FIG. 9. *Brick geometry with a second-order scheme.* Measured wallclock times for the experiments using the Pentium network.

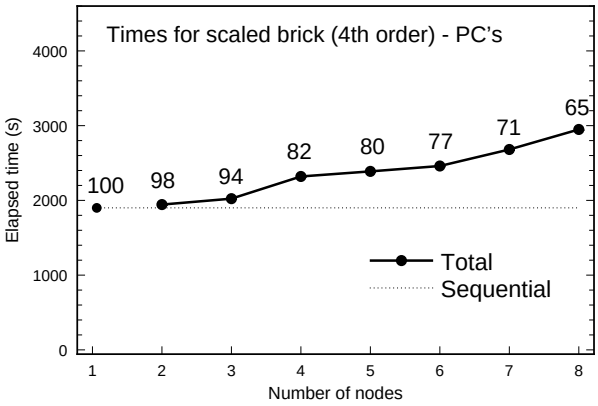

FIG. 10. *Brick geometry with a fourth-order scheme.* Measured wallclock times for the experiments using the Pentium network.

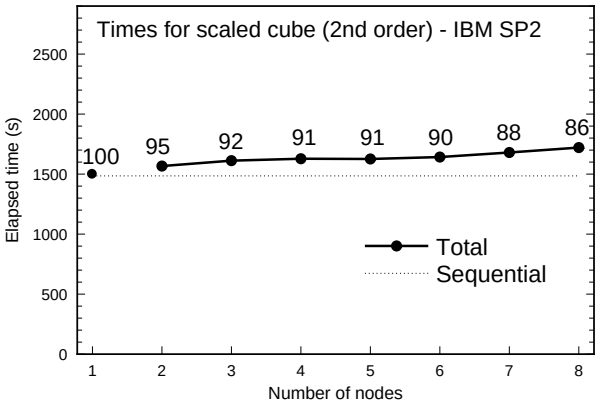

FIG. 11. *Cubic geometry with a second-order scheme.* Measured wallclock times for the experiments using the IBM SP2.

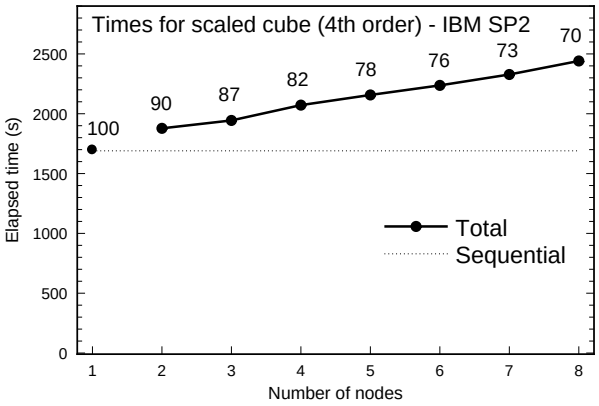

FIG. 12. *Cubic geometry with a fourth-order scheme.* Measured wallclock times for the experiments using the IBM SP2.

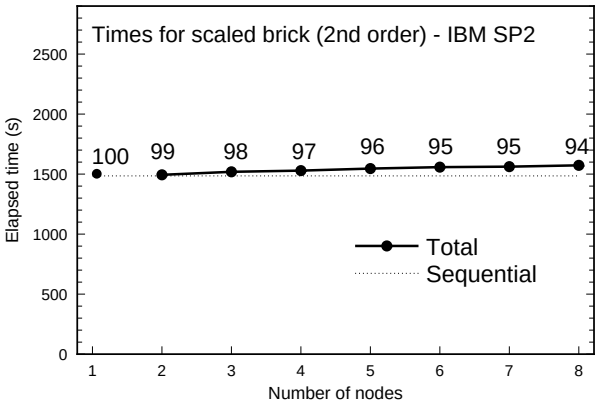

FIG. 13. *Brick geometry with a second-order scheme.* Measured wallclock times for the experiments using the IBM SP2.

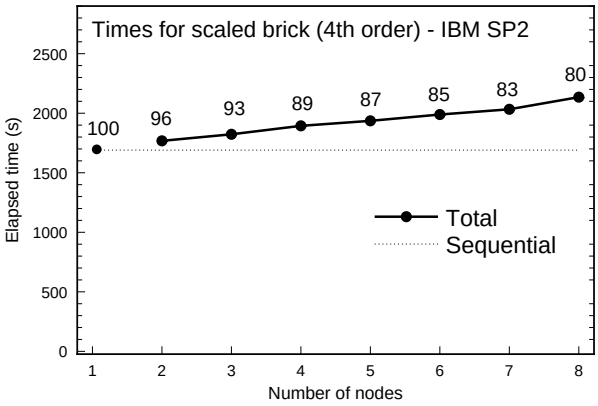

FIG. 14. *Brick geometry with a fourth-order scheme.* Measured wallclock times for the experiments using the IBM SP2.

List of Tables

TABLES

NODES	N_x	N_y	N_z
1	93	93	93
2	116	117	118
3	133	134	135
4	147	147	148
5	158	158	160
6	169	169	168
7	176	176	182
8	184	184	192

TABLE I. *Cubic geometry.* Number of nodes and problem sizes (in grid points) for the experiments in the Pentium network.

NODES	N_x	N_y	N_z
1	74	74	146
2	93	93	186
3	107	106	212
4	117	117	234
5	126	126	252
6	134	134	268
7	141	141	282
8	147	147	294

TABLE II. *Brick geometry.* Number of nodes and problem sizes (in grid points) for the experiments in the Pentium network.

NODES	N_x	N_y	N_z
1	159	159	159
2	200	200	200
3	229	230	228
4	252	252	252
5	270	270	270
6	288	289	288
7	305	305	301
8	320	320	312

TABLE III. *Cubic geometry.* Number of nodes and problem sizes (in grid points) for the experiments in the IBM SP2.

NODES	N_x	N_y	N_z
1	126	126	252
2	159	158	318
3	181	182	363
4	201	201	396
5	216	217	425
6	229	229	456
7	242	243	476
8	252	252	504

TABLE IV. *Brick geometry.* Number of nodes and problem sizes (in grid points) for the experiments in the IBM SP2.