

# *SUDREF*: A distributed elastic reflectivity modeling algorithm

Wences Gouveia

*Department of Geophysics*

*Center for Wave Phenomena*

*Colorado School of Mines, Golden, CO 80401, USA*

## **ABSTRACT**

From earthquake seismology to mineral exploration, accurate seismic modeling is one of the most important tools for studying the Earth's subsurface. Here, I present a distributed implementation of the elastic reflectivity method. In this method, the generation of the synthetic wave field is done by solving a sequence of boundary value problems. Since the computations are performed in the frequency domain, this approach presents a high degree of parallelism.

The distributed implementation (*sudref*) consists of a master-slave topology, in which all the required communication and synchronization procedures are provided by the PVM library. This algorithm is designed to take advantage of a heterogeneous computer network (i.e., with machines of different power or with machines with unequal processor loads), by partitioning the total work into tasks, the size of which is specified by the user.

*Sudref* has been benchmarked on an Ethernet connected network of 11 Pentium-based Linux workstations, a 4 processor SGI Power Challenge and a 16 processor IBM SP2. The distributed implementation shows good performance results. The elastic wavefield for a 400 layer model can be computed with an elapsed time of less than 1 hour on a network of eleven PC's, a speedup of a factor of 8.2 over the sequential time.

**Key words:** Seismic modeling, Distributed computing, Computational performance

## **Introduction**

Due to the relevance of synthetic data generation in studies of the Earth's subsurface, many algorithms have been proposed, developed and tested for modeling waves in heterogeneous elastic media. Those algorithms differ in the complexity of the models they can handle, and also in the extent that the wavefield is modeled (for example, just the primary events as opposed to all models and multiples). Finite difference and finite element modeling techniques, are probably the method of choice if one is interested in synthesizing the complete wavefield in a fully heterogeneous medium. Geometric optics allows rapid computation of the wavefield, and also has the ability to generating single modes of propagation (for example, mode converted *S*-waves). It is however limited to smooth media, or media with large features compared

to the dominant wavelength. In the situation where the medium presents some kind of symmetry, for example the elastic properties do not change laterally, analytic solutions of the wave equation for the complete wavefield, such as reflectivity (Fuchs and Müller, 1971, Kennet, 1983) may be used. This procedure casts the seismic modeling algorithm as the solution of a sequence of boundary value problems, in which the continuity of displacements and tractions across internal interfaces are used to propagate the wavefield from one layer to the next.

Here, I present a distributed implementation of an elastic isotropic reflectivity method called *sudref*. Since in this technique the computations are performed in the offset-temporal frequency domain, and the frequency components are independent of each other, it presents

a high degree of parallelism. Aiming at preserving full portability to different computer systems I developed the code in standard ANSI C and used the PVM library (Geist *et al.*, 1994) for all required synchronization and message passing routines. That allowed me to benchmark this code in the following computer architectures:

- 4 processor SGI Power Challenge
- 1 Mbyte/s Ethernet connected network of 11 Pentium-based Linux workstations
- 16 processor IBM SP2

This work is structured as follows. First, I briefly describe the reflectivity method and illustrate how the algorithm works with an example. Next, I present the distributed implementation of this procedure, and discuss the simple load balancing approach used to make an efficient use of a heterogeneous network. Finally, I show benchmark results in which I measure the speedup of the implementation when several processors are used in the computation.

### The Reflectivity Method

In this section I will introduce the reflectivity method in a schematic way, keeping the usage of mathematical expressions to a minimum. A detailed description of this procedure can be found in the excellent tutorial by Müller (1985). The geometry of the problem in cylindrical coordinates is shown in Figure 1. The elastic isotropic layered medium is completely characterized by the  $P$ -wave ( $\alpha$ ) and  $S$ -wave ( $\beta$ ) velocities, the density ( $\rho$ ) and the thickness ( $dz$ ) for each layer. Consider for a moment a homogeneous medium above and below the source depth  $z_s$ . In this case, the upgoing ( $z < z_s$ ) and downgoing ( $z > z_s$ ) displacement potentials due to a point force located at depth  $z_s$  with orientation  $[F_1, F_2, F_3]$  in Cartesian coordinates are given by (Müller, 1985)

For  $z > z_s$  :

$$\begin{aligned} \Phi_s^d &= \frac{1}{4\pi\rho_m} \int_0^\infty (\epsilon_1 A_{s1} J_0(u\omega r) + \epsilon_2 A_{s2} J_1(u\omega r)) \\ &\quad \exp[-i\omega a_m(z - z_s)] du \\ \Psi_s^d &= \frac{1}{4\pi\rho_m} \int_0^\infty \frac{1}{j\omega} (\epsilon_1 C_{s1} J_0(u\omega r) + \epsilon_2 C_{s2} J_1(u\omega r)) \\ &\quad \exp[-i\omega b_m(z - z_s)] du \\ \chi_s^d &= \frac{1}{4\pi\rho_m} \int_0^\infty \eta E_s J_1(u\omega r) \exp[-i\omega b_m(z - z_s)] du \\ A_{s1} &= u, \quad A_{s2} = \frac{u^2}{ia_m}, \quad C_{s1} = \frac{u^2}{b_m}, \quad C_{s2} = \frac{i}{u}, \\ F_s &= \frac{i}{\beta_m^2 b_m}. \end{aligned} \quad (1)$$

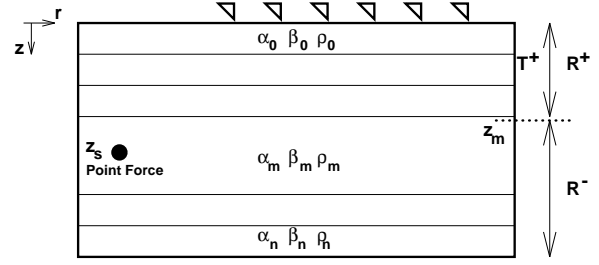


Figure 1. Elastic layered model.

For  $z < z_s$  :

$$\begin{aligned} \Phi_s^u &= \frac{1}{4\pi\rho_m} \int_0^\infty (\epsilon_1 B_{s1} J_0(u\omega r) + \epsilon_2 B_{s2} J_1(u\omega r)) \\ &\quad \exp[+i\omega a_m(z - z_s)] du \\ \Psi_s^u &= \frac{1}{4\pi\rho_m} \int_0^\infty \frac{1}{j\omega} (\epsilon_1 D_{s1} J_0(u\omega r) + \epsilon_2 D_{s2} J_1(u\omega r)) \\ &\quad \exp[+i\omega b_m(z - z_s)] du \\ \chi_s^u &= \frac{1}{4\pi\rho_m} \int_0^\infty \eta F_s J_1(u\omega r) \exp[+i\omega b_m(z - z_s)] du \\ B_{s1} &= -u, \quad B_{s2} = \frac{u^2}{ia_m}, \quad D_{s1} = \frac{u^2}{b_m}, \quad D_{s2} = \frac{i}{u}, \\ F_s &= \frac{i}{\beta_m^2 b_m}. \end{aligned} \quad (2)$$

Here,  $\Phi$ ,  $\Psi$  and  $\chi$  are the scalar potentials in the frequency ( $\omega$ ) domain for  $P$ -waves,  $SV$ -waves and  $SH$ -waves, respectively.  $u$  is the horizontal slowness,  $a_m$  and  $b_m$  are the vertical slownesses of layer  $m$ ,  $\epsilon_1 = F_3$ ,  $\epsilon_2 = F_1 \cos\phi + F_2 \sin\phi$  and  $\eta = -F_1 \sin\phi + F_2 \cos\phi$ , where  $\phi$  is the azimuthal angle.  $J_0()$  and  $J_1()$  are the Bessel functions of the first kind of zero-th- and first-order, respectively. For simplicity, consider only the  $P$ - $SV$  propagation modes. It is convenient to define the following source amplitude vectors

$$S_{1,2}^d = \begin{bmatrix} A_{s1,s2} \\ C_{s1,s2} \end{bmatrix}, \quad S_{1,2}^u = \begin{bmatrix} B_{s1,s2} \\ D_{s1,s2} \end{bmatrix}. \quad (3)$$

The synthesis of the wavefield at the surface requires two steps. First the upgoing wave  $V_{1,2}$  at  $z = z_m$  should be computed. It is not difficult to show (Müller, 1985) that  $V_{1,2}$  are given by

$$V_{1,2} = [I - R^- R^+]^{-1} (S_{1,2}^u + R^- S_{1,2}^d). \quad (4)$$

Here,  $R^-$  and  $R^+$  are the reflectivity matrices for the lower and upper stack of layers, respectively. Such matrices are derived recursively from considerations on only the continuity of the displacement and the traction fields across the interfaces. By the same approach it is possible to compute the transmissivity  $T^+$  of the upper

stack of layers and compute the potential at the surface, which is given by

$$V_{1,2}^0 = T^+ V_{1,2}. \quad (5)$$

The far-field displacement field at the surface is directly derived from the potentials (Aki and Richards, 1980),

$$\begin{bmatrix} u_r \\ u_v \end{bmatrix} (r, \omega) = \omega \sum_{i=1}^2 \epsilon_i \int_0^\infty J_i U V_i^0 du. \quad (6)$$

Here,  $u_r$  and  $u_v$  represent the radial and vertical components of the displacement field, and the following definitions have been used:

$$U = \begin{bmatrix} u & b_0 \\ a_0 & -u \end{bmatrix}, \quad J_1 = \begin{bmatrix} -J_1(u\omega r) & 0 \\ 0 & iJ_0(u\omega r) \end{bmatrix},$$

$$J_2 = \begin{bmatrix} J_0(u\omega r) & 0 \\ 0 & iJ_1(u\omega r) \end{bmatrix}. \quad (7)$$

Equation (6) is used to compute the  $P$ - $SV$  displacement field in the frequency domain. Of course, each frequency component of the wavefield is independent of each other, a feature that makes the reflectivity method a natural algorithm for parallel and distributed implementations.

### Example

An elastic reflectivity code requires that the user supplies the thickness,  $P$ -wave and  $S$ -wave velocities and density, for each layer. Also, for this specific implementation, the quality factor  $Q$  for each layer is also required, one for  $P$ -waves and another one, possibly different, for  $S$ -waves. Figure (2) illustrates depth profiles obtained from blocking well-logs acquired at the Sorrento field, south-east of Colorado. There are approximately 400 layers in this model. Using those profiles and a constant quality factor of 1000 for both  $P$ - and  $S$ -waves, I generated the synthetic shot gather shown in Figure (3). This is the vertical component of the displacement field acquired at the surface due to a vertical point source sitting at the same location of the first receiver of this gather. The computation of these seismograms took around 8 hours on a 90 MHz Pentium-based personal computer. In the next section, I describe the distributed implementation of this technique.

### SUDREF: A Distributed Implementation of the Reflectivity Method

The distributed implementation of the reflectivity method is based on the simple idea of a master processor assigning frequency partitions to slave processors. Figure (4) illustrates the idea. As the slave processors

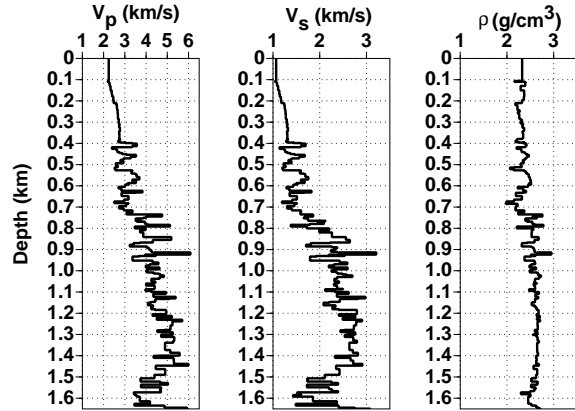


Figure 2.  $P$ -,  $S$ -wave velocity and density profiles.

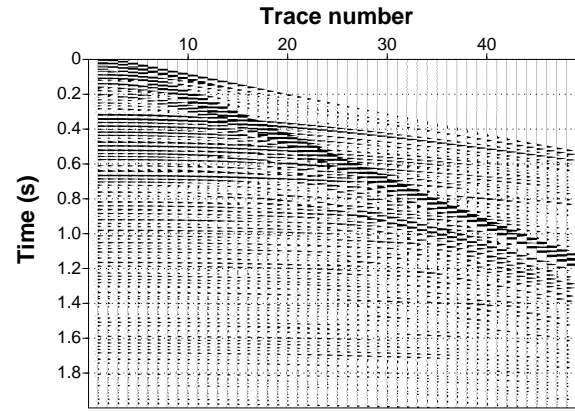
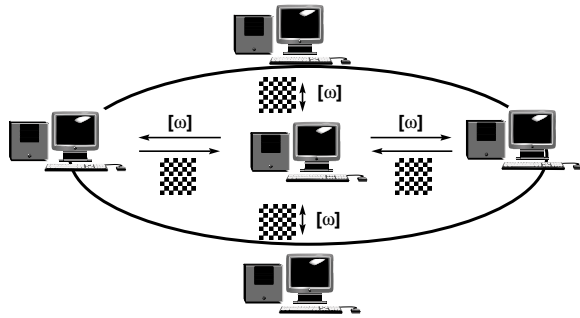
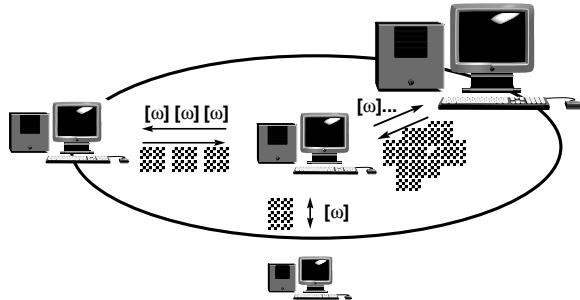


Figure 3. Vertical component of the displacements generated for the model in Figure (2).

finish the computation of their respective frequency partitions, they send the results back to the master processor. As a final step the master processor transforms the data to the time domain. As mentioned before, *sudref* is implemented in standard ANSI C, and all the message passing and synchronization routines are provided by the PVM library. The implementation depicted in Figure (4) is probably close to an optimum one in the case of a homogeneous network of computers, on which the only running program is the distributed reflectivity algorithm. This is not usually the case. The typical scenario is to have computers with distinct computational power and/or with different levels of CPU utilization connected to the same network. This characterizes a heterogeneous computer environment. In this situation some strategy of load balancing is required in order to allocate more work to computers of higher computational capacity. The load balancing used here is simple, but it provided satisfactory results. It consists of splitting the total modeling job



**Figure 4.** The distributed reflectivity algorithm.  $[\omega]$  indicates the frequency interval assigned to each slave processor by the master. The “ball” represents the frequency components of the displacements computed by the slave processors.

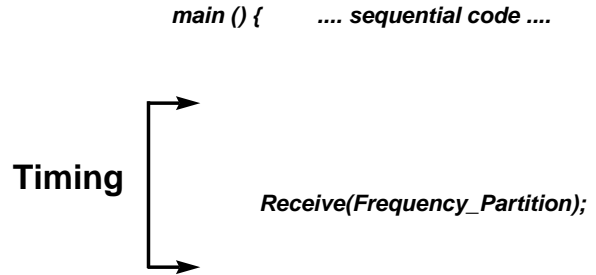


**Figure 5.** The load balancing strategy of *sudref*.

into small tasks, the size of which (the number of frequencies to be computed) is specified by the user. Once the slave processors finish their computations for a specific frequency partition, they return their data to the master processor. If there are still more frequency components to be computed the master processor sends one more task to the slave processor. This procedure is repeated until no more tasks are left. In this implementation, more powerful machines will tend to receive more tasks than less powerful ones. Figure (5) illustrates how *sudref* works. In the next section I show benchmark results of the distributed implementation for shared- and distributed-memory computer architectures. The measured speedups tend to deviate from the linear as more processors are used in the computation of the synthetic seismograms. Nonetheless, as expected due to the high level of parallelism of the reflectivity algorithm, this technique is very suitable for parallel and distributed implementations.

**Performance Measurements**

The computational performance of *sudref* was evaluated on shared- and distributed-memory computer architectures, via speedup measurements. During the bench-



**Figure 6.** Timing scheme for benchmarking of *sudref*.

marking the computer was completely dedicated to this application. The test model consists of approximately 400 layers and it is shown in Figure (2). The benchmarking procedure consists of measuring the elapsed time of the frequency loop and normalizing it by the elapsed time of the sequential run (see Figure 6). Notice that, since the frequency loop is the one executed in a distributed fashion, the *sequential* portion of the algorithm was not considered in the speedup measurements. This sequential portion consists basically of the transformation of the synthetic displacements from frequency to time domain, which represents a small fraction of the total execution time.

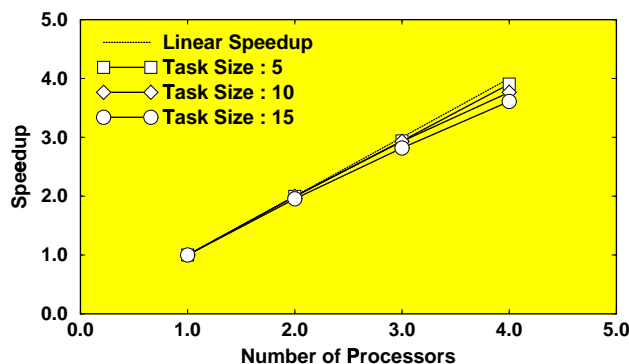
All the measurements were performed for 3 distinct task sizes. As mentioned before the size of the task is the number of frequency components that the master processor assigns to the slave processor at each time. A larger task size implies a smaller number of interactions (through message transfers) between master and slave, but also implies a larger message to be sent from slave to master. In the experiments presented here the size of this message ranges from 1.9 Kbytes (task size = 5 frequencies) to 5.6 Kbytes (task size = 15 frequencies). The task size can be regarded as a tuning parameter that can be used to yield a better performance of *sudref* on a specific hardware. The speedup measurements are discussed next, with a brief description of the computer architecture used in each one of the benchmarks.

**Silicon Graphics Power Challenge**

The basic configuration of this machine is described in Table (1). Figure (7) shows the speedup curves for *sudref* on the Power Challenge. The speedup is very close to linear. For this problem, slightly better performance results were obtained for a task size of 5.

Architecture:	Shared-memory
Operating System:	Irix v6.1
Processor:	MIPS - R8000
Number of Processors:	4
Clock Speed:	75 MHz
Real Memory per Processor:	64 Mbytes

**Table 1.** Silicon Graphics Power Challenge basic configuration.



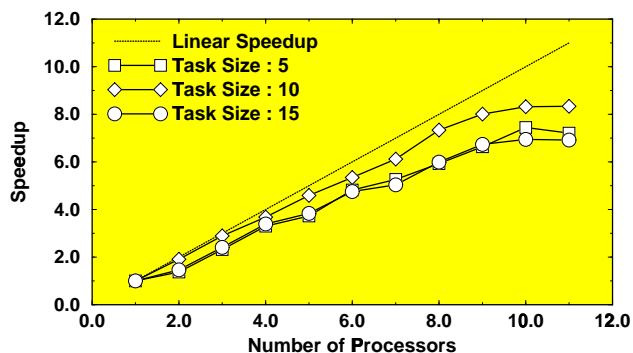
**Figure 7.** Performance results for the Silicon Graphics Power Challenge.

### Pentium-Based Network of Linux Workstations

Here, I benchmarked *sudref* on a heterogeneous (due to different clock speeds) network of Pentium processors. Its basic configuration is described in Table (2). Figure (8) shows the speedup curves for *sudref* on this platform, which were obtained by averaging several runs. Alternative measurement procedures are of course possible for this case. When more than 4 processors are used in the computation, the speedup begins to depart from linear. Since the size of the messages exchanged by all processors do not exceed the bandwidth of the Ethernet, this reduction in performance is attributed to the larger overhead

Architecture:	Distributed-memory
Operating System:	Linux v1.2+
Processor:	Intel Pentium
Number of Processors:	11
Connection:	Ethernet (1 Mbyte/s)
Clock Speed:	90 - 120 MHz
Real Memory per Processor:	32 Mbytes

**Table 2.** Pentium-based network basic configuration.



**Figure 8.** Performance results for the Pentium-based network.

Architecture:	Distributed-memory
Operating System:	AIX v4.0
Processor:	IBM RS6000/591
Number of Processors:	16
Connection:	Switch-based (40 Mbyte/s)
Clock Speed:	66.7 MHz
Real Memory per Processor:	512 Mbytes

**Table 3.** Pentium-based network basic configuration.

imposed to the master processor to manage a larger number of tasks. Recall that the slave processors remain idle while waiting for the frequency range assigned by the master processor. As the number of processors increase, so does this waiting time. Therefore the computational performance degrades. In this example the best speedups were obtained with a task size of 10.

### IBM SP-2

Finally, I evaluated the performance of *sudref* on an 16-processor IBM SP2. Table (3) describes the basic configuration of this system. Due to the higher bandwidth provided by this system, as compared to the previous one, I was able to obtain speedup results very close to linear when up to 11 processors were used (see Figure (9)). After that the computational performance of the distributed code begins to degrade. Since we are far from exceeding the bandwidth of the high performance switch that connects the processors, the same comments made in the previous section apply here. When more processors are added to the computation, the overlap between communication and computation is smaller, reducing the overall performance of the application. Note

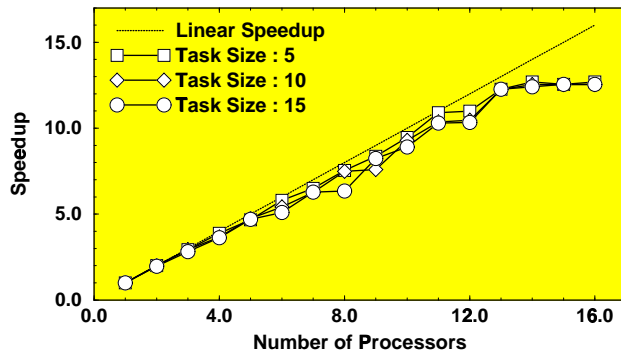


Figure 9. Performance results for the IBM SP2.

that in this case the results were independent of the task size.

### How to get the code

The source code and installation instructions for *sudref* can be downloaded from the CWP Linux WWW server (<http://landau.Mines.edu/>). *Sudref* is written in ANSI-C and uses the PVM library version 3.3.x.

### Conclusions

In this work I presented a distributed implementation of an elastic isotropic reflectivity code. Since the modeling of synthetic seismograms is performed in the frequency domain, this procedure is very attractive for parallel and distributed implementations. The distributed reflectivity code was benchmarked on 3 computer platforms, including shared- and distributed-memory systems. On a network of 11 PC's, the time required to solve a 400 layer problem dropped from 8 hours to less than 1 hour, a speedup higher than 8 over the sequential time. Improvements on those results are possible, by increasing the overlap between communication and computation.

Finally, the fact that *sudref* can be used on any computer platform that supports the C programming language and PVM makes the program very attractive for the computation of synthetic seismograms for a large number of layers.

### Acknowledgments

I thank Prof. John Scales for the constant encouragement. Discussions with Alejandro Murillo and Alberto Villarreal on benchmarking and PVM tricks were very

helpful. I am grateful to the IBM POWER Parallel Systems remote access program, which provided the dedicated time on the SP-2, and to the Reservoir Characterization Project for the well-log data used to build the test model for the benchmarks.

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 Army Research Office and the National Science Foundation under grant DMS-9506603.

### References

- Fuchs, K. and Müller, G. 1971. Computation of synthetic seismograms with the reflectivity method and comparison with observations. *Geophys. J.R. Astron. Soc.*, **11**, 417–433.
- Kennet, B. 1983. *Seismic wave propagation in stratified media*. Cambridge University Press.
- Müller, G., 1985. The reflectivity method: a tutorial. *J. Geophysics*, **58**, 153–174.
- Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. & Sunderam, V. 1994. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press.