A Parallel 3D Parabolic Wave Equation Solver

Ding Lee[1], Diana C. Resasco, Martin H. Schultz[2]
Faisal Saied[3]

YALEU/DCS/RR-995
December 1993

The authors were supported in part by the Office of Naval Research (ONR) under contracts N00014-89-J-1671 and N00014-93-WX-24092, by the Naval Undersea Warfare Center (NUWC) independent research project A10003 and by grants from IBM and NSF ASC 92 09502 RIA.

[1]Naval Undersea Warfare Center
[2]Yale University, Dept. of Computer Science
[3]Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science

Abstract

Three dimensional (3D) models of sound propagation in the ocean can lead to very large scale computations. With the advent of parallel computing, we have the chance of doing these computations at an acceptably fast rate.

We describe our work towards porting a 3D parabolic equation solver to current parallel computers. The code is FOR3D, developed at NUWC, and is based on the Lee-Saad-Schultz model. This model takes the azimuthal coupling into account, and marches the solution out in range, with an AD scheme, which requires alternated sweeps, solving independent tridiagonal systems alternating in the depth and azimuth directions at every range step.

Our parallel implementation is in Linda, a language that allows parallel algorithms to be expressed in terms of a machine-independent model of parallel computing. Codes parallelized in Linda will run on any parallel computer on which Linda is implemented. We will focus on a cluster of workstations viewed as a distributed memory multiprocessor.

We report on the improvement in performance that can be obtained through a combination of algorithmic improvements and parallel computing. We touch upon some software engineering issues that have a strong bearing on the process of porting existing codes to parallel architectures.

We demonstrate that workstation clusters can be very effective for scientific codes, particularly when a fast interconnect or switch is used.

1 Introduction

In this paper, we describe our progress in implementing a portable parallel version of FOR3D, a code developed at the Naval Undersea Warfare Center (NUWC) for the prediction of sound propagation in the ocean. Section 2 describes the equations solved by the original FOR3D code. In Section 3 we discuss some general considerations on implementing a parallel version of the code. A brief description of Linda is given in Section 4 and the hardware used for our experiments is described in Section 5. In Section 7 we show performance data, we discuss possible future directions of our research in Section 8, and in Section 9 we present our conclusions.

2 Description of the Problem and Numerical Approach

FOR3D solves the following parabolic equation, which models one-way outgoing propagation:

$$u_r = ik_0(-1+\sqrt{1+X+Y})u \quad , \tag{1}$$

where u is a function of depth (z), azimuth (θ) and range (r), k_0 is a constant, the reference wavenumber, and the differential operators X and Y are given by

$$\begin{aligned} X &= n^2(r,\theta,z) - 1 + \frac{1}{k_0^2}\rho\frac{\partial}{\partial z}(\frac{1}{\rho}\frac{\partial}{\partial z}) \\ Y &= \frac{1}{k_0^2 r^2}\rho\frac{\partial}{\partial \theta}(\frac{1}{\rho}\frac{\partial}{\partial \theta}) \end{aligned} \tag{2}$$

where $n(r,\theta,z)$ is the index of refraction and ρ is the density. In FOR3D, ρ can be a step-wise linear function of z and θ [5].

A local solution to (1) can be written down symbolically as:

$$u(r+\Delta r,\theta,z) = e^{-ik_0\Delta r}e^{ik_0\Delta r\sqrt{1+X+Y}}u(r,\theta,z). \tag{3}$$

The numerical scheme designed by Lee, Saad and Schultz [4] makes use of the following rational approximation to the square root operator:

$$\sqrt{1+X+Y} \approx 1 + \frac{1}{2}X - \frac{1}{8}X^2 + \frac{1}{2}Y.$$

By assuming near commutativity of the operators X and Y, (3) becomes:

$$u(r+\Delta r,\theta,z) = e^{-ik_0\Delta r}e^{ik_0\Delta r(1+\frac{1}{2}X-\frac{1}{8}X^2)}e^{ik_0\Delta r\frac{1}{2}Y}u(r,\theta,z).$$

Then, the exponential operators are further approximated by rational functions, giving the following marching scheme:

$$u(r+\Delta r,\theta,z) = \frac{I+\alpha X}{I+\bar{\alpha}X}\quad\frac{I+\beta Y}{I+\bar{\beta}Y}\quad u(r,\theta,z)\quad , \tag{4}$$

where $\alpha = \frac{1}{4} + i\frac{1}{4}k_0\Delta r$ and $\beta = i\frac{1}{4}k_0\Delta r$.

A finite difference discretization of (4) leads to a problem of the form

$$ABU^+ = A^*B^*U \tag{5}$$

where U and U^+ are the computed solution vector at the present range r and at $r+\Delta r$, respectively, and A and B represent the discretizations of the operators $I+\alpha X$ and $I+\beta Y$ respectively, computed at range $r+\frac{1}{2}\Delta r$.

The computational domain at each range step is an N_z by N_θ grid, originally numbered column-wise, i.e., by depth first, then by sector. With such ordering, the matrix A is block-diagonal, with tridiagonal blocks in the diagonal. Similarly, if the gridpoints are numbered by sector first (i.e., row-wise), then the matrix B is block-diagonal, with tridiagonal blocks in the diagonal.

At each range step, computations are arranged in two AD half-steps, alternated with a transposition (reordering) of the partial solution vector:

- **z-half-step**: Compute right hand side and solve
$$A\tilde{U} = A^*B^*U \tag{6}$$
- **Transpose** intermediate solution $\tilde{U}$
- **θ-step**: Solve
$$BU^+ = \tilde{U} \tag{7}$$
- **Transpose** new solution U^+ back to original ordering.

The FOR3D code is described in detail in [1].

3 Parallel Implementation: Some General Considerations

In this section we briefly outline some general features of our approach to parallelizing FOR3D, independent of any particular parallel hardware/software systems. In the subsequent sections we will describe how this approach is implemented on clusters of workstations using LINDA.

The basic structure of a range step in FOR3D consists of a set of independent tridiagonal solves in depth followed by another set of independent tridiagonal solves in the azimuthal direction. In addition, matrix vector products involving

tridiagonal matrices are required in each of these directions to form the right hand side.

Our approach to parallelizing these computations is to assign one or more of the (depth dependent) tridiagonal systems to each processor. Each processor forms the tridiagonal matrices it needs, and hence the matrices are not involved in the communications. The solves are done locally, using Gaussian elimination adapted to the structure of the matrices.

In the second phase of the range step, each processor solves one or more tridiagonal systems in the θ direction. The matrices for this phase are again formed locally, and are not involved in any inter-processor communication. However the right hand sides of the tridiagonal systems in this phase depend on the intermediate solution obtained from the first half-step, and data movement is required between these phases.

This data movement can be conceptually viewed as matrix transposition and can be implemented in several different ways. The simplest approach involves each processor sending data to every other processor and receiving data from every other. The global nature of the communication reflects the global data dependencies inherent in the FOR3D model. Any 3D model that takes full θ-coupling into account will have analogous global communication requirements. It is important to exploit any features of the parallel hardware and software to make this phase as efficient as possible to minimize the overhead cost of parallelization.

Because of the well-structured nature of the computational kernel in FOR3D, we achieve good load balance across processors.

4 Linda

Linda is a coordination language that complements traditional languages for computation. In our application, we used Fortran-Linda from Scientific Computing Associates. A few simple commands are added to Fortran. The resulting language is architecture-independent, which makes the code highly portable.

The Linda coordination model is based on a form of virtual shared memory, called "tuple space", that is designed specifically to accommodate inter-process coordination. Linda-style shared memory has been efficiently implemented in settings such as distributed-memory parallel machines and local area networks, whose architectures preclude communication via conventional shared memory. A tuple space stores tuples, which are ordered aggregates of data objects. Linda provides three basic access operations with built-in synchronization: the `out` operation generates a tuple and adds it to memory; the `in` operation looks for some "matching" tuple and removes it, blocking if necessary until one is available; the `rd` operation is like `in`, but copies rather than removes the matched tuple. Tuple space is an associative memory: `in` or `rd` statements specify a "matching template" or anti-tuple which may include either values or typed place-holders

or both. Linda also provides a process-creation mechanism integrated with the tuple space abstraction: the **eval** operation generates and places in tuple space an unevaluated tuple. When each field of the unevaluated tuple has been fully evaluated, the unevaluated tuple turns into an ordinary tuple which can be read or removed using the standard operations.

For a more detailed description of Linda and its applications, see [2].

5 Hardware

For our experiments, we used a network of IBM RS6000/560 workstations. With the Ethernet interconnect, only one processor pair can communicate at a time. A faster interconnect was recently added, IBM's AllNode (or V7), a switch which supports multiple, high-bandwidth, low-latency connections between processors.

6 Parallel FOR3D

In our parallel version of FOR3D, the computational domain is initially distributed so that each processor gets a sector (i.e. a number of azimuth values). In homogeneous networks, each processor works on a data set of approximately the same size, but the code accepts data sets of different sizes, to provide flexibility for good load balance when working on heterogeneous networks. For each range step, the computations in the depth direction are performed locally, then the partial solution is transposed across processors, so that each processor can locally compute the azimuth step, and then the solution is transposed back.

Instead of approximating 4 by $U^+ = B^{-1}A^{-1}A^*B^*U$ (solved by steps (6) and (7) in section 2), we use the ordering $U^+ = B^{-1}B^*A^{-1}A^*U$. This ordering of the operators facilitates the organization of the computation, because it permits us to group the computations in each alternated direction. The two transpose operations at every range step make sure that the data for each half-step is local to each processor, therefore these steps are performed with perfect parallel speed-up. All communication between processors is done at the transpose phase.

- z-**half-step**: Compute right hand side and solve (Each processor computes a portion of the solution.)

$$A\tilde{U} = A^* U \tag{8}$$

- **Transpose** intermediate solution $\tilde{U}$
- θ-**step**: Compute right hand side and solve (Each processor computes a portion of the solution.)

$$BU^+ = B^* \tilde{U} \tag{9}$$

- **Transpose** new solution U^+ back to original ordering.

Master-worker model

Linda allows the user to design a parallel application using the master-worker model, in which one of the processors acts as master, assigning data and tasks to the spawn processes. Although it is not essential, we chose to use this model for our parallel FOR3D. The master process is in charge of processing and distributing the environmental data, and for gathering output for visualizing the solution when requested by the user. It is possible to program the master processor to become a worker in cases when it is likely to stay idle for long periods of time.

Fig. 1 illustrates the sequence of computation and communication at a particular set of range steps, for a case of a master process and three workers. The arrows represent data being communicated among processors. We need to point out that this picture is an oversimplification of the communication pattern, and it does not necessarily represent the Linda model. Linda preprocesses the communication pattern in order to minimize overhead.

The master process gathers the solution vector for visualization when necessary (in most applications, the solution is not printed out at every range step), while the workers can continue marching the computations in range. When new environmental data needs to be inputed (again, this is not usually done at every range step), it is the master processor's job to process the new data and distribute it among the workers.

7 Parallel Performance

In the current implementation, the transpose step is the communication bottleneck. The cost for the transpose step increases linearly with the volume of data being transposed. Since we get perfect speed-up for the arithmetic computations, the amount of speed-up that can be achieved for a given problem size, as we increase the number of workers (processors), is basically bounded by the cost of the transpose. In Fig. 2 we show run time curves for two problem sizes,

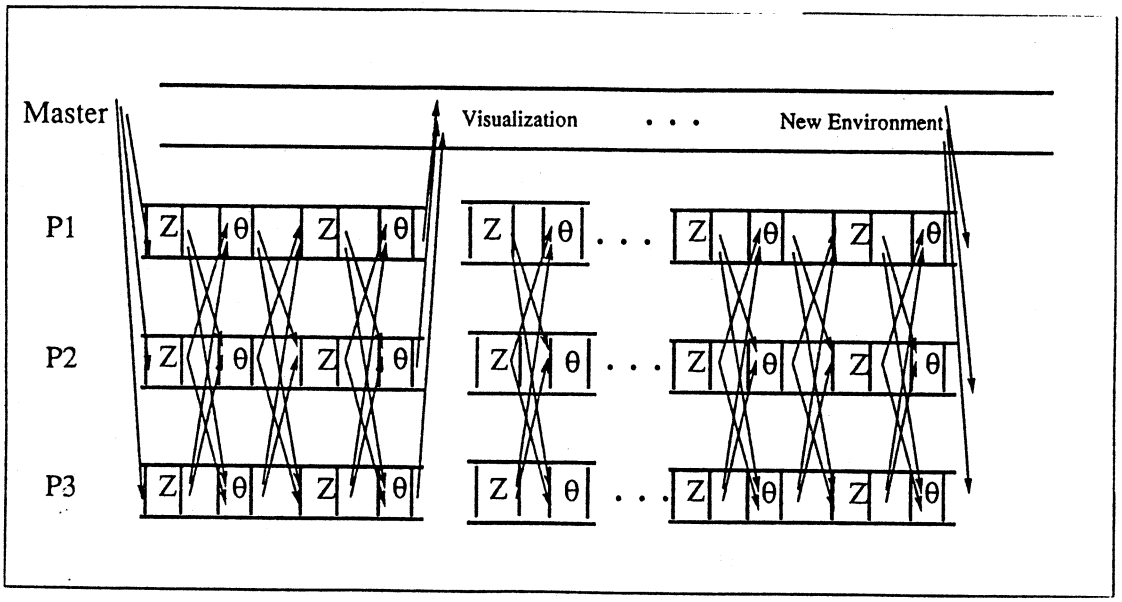

Figure 1: Communication and computation pattern

Table 1: Runtimes for $3D$ and $N \times 2D$ models
Run-time for one range step computation.
Grid size N_z=799 $\times$ N_θ=180

Number of Processors	3D	N×2D
1	4.20	3.20
2	2.80	1.61
3	1.97	1.12
4	1.62	0.85

for increasing number of processors. The solid lines indicate perfect speed-up (i.e. the one-processor time divided by number of processors). We can see that arithmetic times agree with the perfect speed-up curve, but as more workers are added, the time for the transpose dominates, making the parallel code less efficient. Fig. 3 shows speed-up curves for the same problems. These results are obviously heavily dependent on architectural features of the network or parallel machine. In Fig. 4 we can see the substantial improvement in efficiency achieved by the same processors when going from an Ethernet interconnect to the faster AllNode switch interconnect.

When θ coupling can be ignored, FOR3D can solve the equations in $N \times 2D$ mode, i.e. as a collection of independent two-dimensional problems. Basically, the θ-half-step is skipped. In this case, the code is not only faster, but obviously more parallelizable, since no transpose is needed. Table 1 shows runtimes per range step for a particular gridsize. We can see how the times for $N \times 2D$ follow perfect speed-up closely.

Finally, as an example of the improvement in performance that can be obtained through a combination of algorithmic and hardware improvements, code optimization, and parallel computing, we show in Table 2 the progression in computing time from a run of the original FOR3D code on a Sparc II workstation to our parallel version on five IBM RS6000/560 workstations.

8 Future Directions

In order to make FOR3D more efficient, we envision improvements to the sequential as well as the parallel versions.

One such improvement to the sequential code will be to replace

$$(I + \bar{\alpha}X)^{-1}\,(I + \alpha X) = \frac{\alpha}{\bar{\alpha}} I + (1 - \frac{\alpha}{\bar{\alpha}})(I + \bar{\alpha}X)^{-1}$$

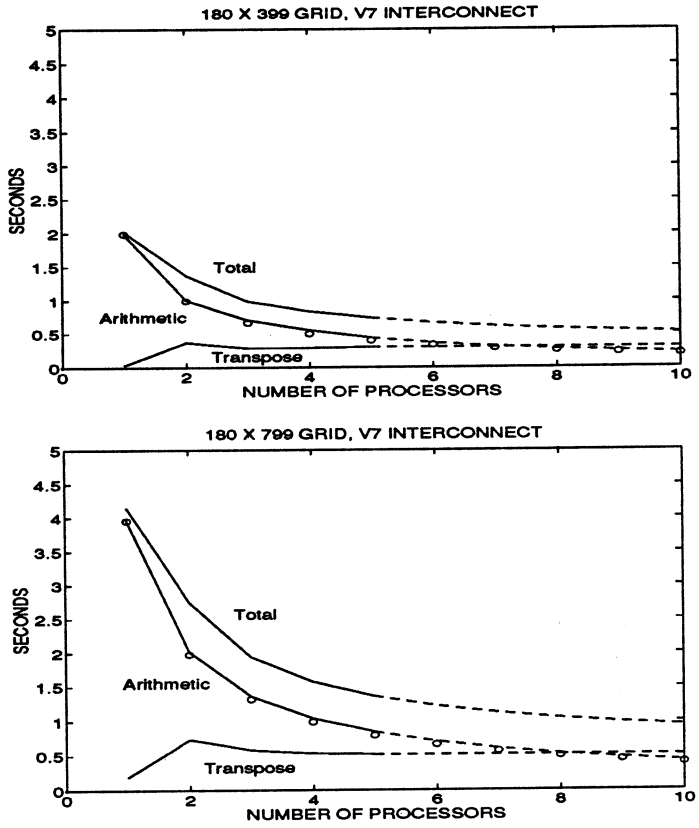

Figure 2: Parallel Performance of FOR3D
Run-time for two grid-sizes, as a function of the number of processors. Total time is the combination of transpose time plus arithmetic time. Dashed lines represent extrapolated values.

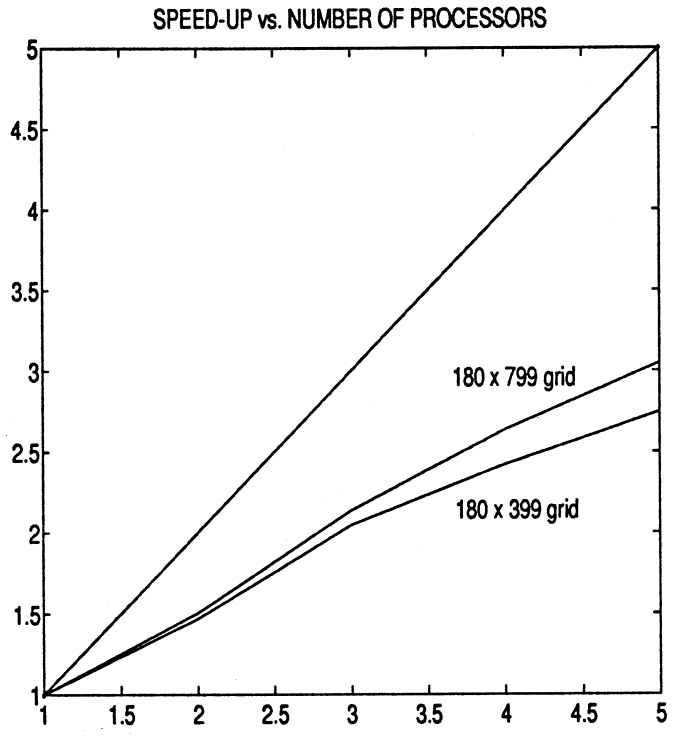

Figure 3: Parallel Performance of FOR3D: Speed-up

and

$$(I+\bar{\beta}Y)^{-1}\,(I+\beta Y) = \frac{\beta}{\bar{\beta}}I + (1-\frac{\beta}{\bar{\beta}})(I+\bar{\beta}Y)^{-1} = -I + 2(I+\bar{\beta}Y)^{-1}$$

in (4). This substitution eliminates the need to compute the right hand sides in the z and θ half-steps. We will report more on this at a later time.

In order to reduce the number of transpose steps needed, another idea is to use alternating sweep ordering, in which the (z and θ) half-steps are followed by (θ and then z) half-steps, thus eliminating the need for one of the two transposes per range step.

Since the cost for the transpose depends on the volume of data transfered, one variation of the algorithm is the use of substructuring (reduced system) techniques to solve the tridiagonal systems for the θ half-step. Instead of transposing the whole array, partial Gaussian elimination is applied to sections of

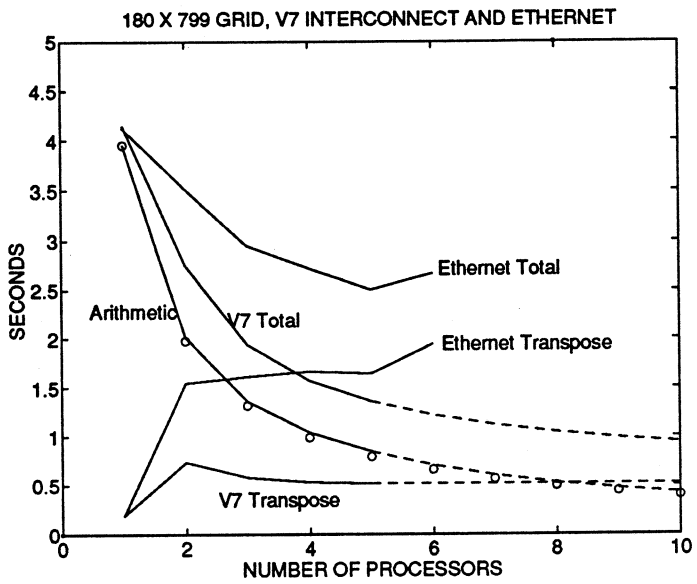

Figure 4: Comparison between Ethernet and AllNode interconnections

each system, and only a small reduced system is exchanged and solved through the network. This method doubles the operation count for the tridiagonal solves, but in most cases it will greately reduce the cost for the transpose, because the volume of data being transfered is reduced to the size of a separator set (about N_z vs. $\frac{N_z \times N_\theta}{p}$ per processor) [6, 3, 7, 8].

We finally point out that more parallelism is to be obtained in real applications by simultaneously solving for several frequencies. The computation for each frequency is completely independent.

9 Summary and Conclusion

We have described the implementation of a functioning portable parallel code for computational ocean acoustics. The tremendous advances in workstation technology have made the workstation cluster approach to parallel computing very attractive. Our experiments on a cluster of high-end workstations with a fast interconnect show that high performance can be achieved in a cost effective manner.

We used the coordination language Linda for our implementation. Linda is easy to use and has low overhead. Code written with Linda will run in any

Table 2: Speed improvement
Run-time for one range step computation.
Grid size $N_z = 799, N_\theta = 180$

Sparc II	
Original (sequential) code	49.0 sec
New (sequential) code	24.2 sec

IBM *RS*6000/560	
New (sequential) code	4.2 sec
Parallel code: 5 procs.	1.3 sec

parallel computer that supports Linda.

The experience gained by designing this application will be useful in parallelizing other similar acoustic codes.

10 Acknowledgements

This research was supported in part by the Office of Naval Research (ONR) under contracts N00014-89-J-1671 and N00014-93-WX-24092, by the Naval Undersea Warfare Center (NUWC) independent research project A10003 and by grants from IBM and NSF ASC 92 09502 RIA.

References

[1] G. Botseas, D. Lee, and D. King. FOR3D: A computer model for solving the LSS three-dimensional wide angle wave equation. Technical Report TR# 7943, Naval Underwater Systems Center, 1987.

[2] N. Carriero and D. Gelernter. *How to write Parallel Programs: A First Course.* MIT Press, Cambridge, MA, 1990.

[3] S. L. Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. and Stat. Comput.*, 8:(354/392), 1987.

[4] D. Lee, Y. Saad, and M. H. Schultz. An efficient method for solving the three-dimensional wide angle wave equation. In *Computational Acoustics, Vol 1: Wave Propagation*, Amsterdam, 1988. North-Holland.

[5] D. Lee, Y. Saad, and M. H. Schultz. A three-dimensional wide angle wave equation with vertical density variations. In *Computational Acoustics:*

Ocean-Acoustic Models and Supercomputing, pages 143–154., Amsterdam, 1990. North-Holland.

[6] F. Saied. *Numerical techniques for the Solution of the Time-dependent Schrödinger Equation and their Parallel Implementation.* PhD thesis, Yale University, 1990. Available as Research Report YALEU/DCS/RR-811, Department of Computer Science, Yale University.

[7] A. Sameh and D. Kuck. On stable parallel linear system solvers. *J. ACM*, 25:(81/91), 1978.

[8] H. H. Wang. A parallel method for tridiagonal equations. *J. ACM Trans. Math. Softw.*, 7:(170/182), 1981.