J. Ind. Geophys. Union (2003)
Vol.7, No.1, pp.11-14

Parallelization Strategies for Seismic Modeling Algorithms

Subrata Chakraborty, Sudhakar Yerneni, Suhas Phadke[1] and Dheeraj Bhardwaj[2]

Centre for Development of Advanced Computing, Pune University Campus, Pune – 411 007
[1]WesternGeco, Houston, TX USA
[2]Indian Institute of Technology, Delhi

ABSTRACT

Parallelism is the key to performance of seismic processing algorithms on any system manufactured today. This article describes the various strategic issues related to the development of parallel acoustic and elastic wave modeling algorithms. It also gives an insight into MPI (Massage Passing Interface) implementation on a parallel computer. Performance analysis is carried out to find the best strategies to develop scalable parallel algorithms for large problem size on large number of processors.

INTRODUCTION

Seismic Modeling is an area of significant research in Industry. Wave equation modeling is useful for understanding complex imaging problems such as shadow zones, steep dips, gas clouds, artifacts etc., for processing and algorithm testing, and for AVO analysis. Thus, the main purpose of forward modeling is to validate the geological model by comparing the synthetic data with the field data.

In the last two decades the power of computers has increased approximately by 500 times and network speed has increased to several thousand times. But on the other side applications like Seismic modeling, need computational resources far greater than a present sequential computer can provide. Parallel processing has proven to be a viable solution to improve performance, which uses the power of multiple computers connected together by high-speed network.

Forward modeling, where the synthetic data is generated for a given earth's model, is key step in process of seismic inversion, where one tries to estimate the physical properties of the earth. 80 to 90 % of the computing time in an inversion algorithm is spent on generating synthetic data. Efficient parallel algorithms are therefore essential.

In this paper, we have discussed some of the strategies for solving efficiently the 3D acoustic and elastic wave equations for seismic modeling on parallel computers.

MATHEMATICAL PROBLEM

The basic problem in the theoretical seismology is to determine the wave response of a given model to the excitation of an impulse source by solving the wave equations under some simplifications. In scalar approximation, the acoustic wave equation may be solved to evaluate the waveform but only compressional waves are considered. A more complete approach is to study the vector displacement field using full elastic wave equation for modeling both, compressional and shear waves. In this formulation the mode conversions are automatically accounted for.

ELASTIC AND ACOUSTIC WAVE MODELING

The mathematical model for elastic wave propagation in heterogeneous media consists of coupled second order partial differential equations governing motions in x-, y- and z-directions. Instead of solving second-order coupled partial differential equations (PDE), we formulate them as a first order hyperbolic system (Virieux 1986; Dai, Vafidis & Kanasewich 1996). First order hyperbolic PDE formulation does not contain any derivatives of physical parameters (Phadke, Bhardwaj & Yerneni 2000). Thus, we need not calculate the gradients of the physical parameters that may cause singularity in the numerical solution due to sharp changes in the subsurface properties.

When we move from elastic to acoustic media, the value of μ becomes zero. By substituting $\mu = 0$ in the elastic formulation, we get a first order system of hyperbolic partial differential equations which governs the acoustic wave propagation (Phadke, Bhardwaj & Yerneni 2000).

The mathematical equations for acoustic wave propagation in 3D homogeneous media can be written as a first order hyperbolic system of partial differential equations:

$$\frac{\partial P}{\partial t} = A\frac{\partial P}{\partial x} + B\frac{\partial P}{\partial y} + C\frac{\partial P}{\partial z} \qquad (1)$$

$$\mathbf{P} = \begin{bmatrix} p \\ u \\ v \\ w \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 0 & \lambda & 0 & 0 \\ \rho^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 \\ \rho^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & \lambda \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \rho^{-1} & 0 & 0 & 0 \end{bmatrix}$$

For solving above partial differential equations, we use finite difference methods.

NUMERICAL SULUTION

Finite difference methods have been preferred for determining the numerical solution of wave equations as they not only account for direct waves, primary reflected waves and multiply reflected waves, but also for head waves, diffracted waves, critically refracted waves observed in ray theoretic shadow zones.

Explicit finite difference methods are most common for solving hyperbolic system of equations. We use the method of splitting in time. An explicit method based on MacCormack scheme is used for numerical solution (Mitchell & Griffiths 1981).

BOUNDARY CONDITIONS

Since a digital computer has finite memory capabilities, we have to restrict the model size to a fixed number of grid points. This introduces artificial boundaries at the edges of the model. In reality the earth is infinite and therefore all the energy impinging on these boundaries must be absorbed. For the finite difference scheme presented here a sponge boundary condition as described by Sochacki et al. 1987, is used for attenuating the energy impinging on the left, right, front, back and bottom edges of the model. To implement sponge boundary condition extra grid points are added to gradually attenuate the energy. The free-surface condition is applied to the top boundary.

ACCURACY, STABILITY AND GRID DISPERSION

The MacCormack method is fourth order accurate in space and second order in time. The model discretization is based upon regular grid. Keeping the grid spacing smaller than one tenth of the shortest wavelength minimizes grid dispersion.

The McCormack finite difference method is stable if

$$\Delta t \leq \frac{\min(\Delta x, \Delta y, \Delta z)}{\sqrt{2}V_{\max}} \qquad (2)$$

where ($V=K/\rho$) and V_{max} is the maximum wave velocity in the medium..

In order to avoid grid dispersion, we have to discretize the model with fine grid spacing. Such discretizations substantially increase the computational size of the model in terms of the number of grid points in each spatial direction and therefore need large computer memory for computation and storage. For explicit schemes, the stability and grid dispersion conditions restrict the size of the time step, which is normally very small. Therefore, the computational time required for calculations become very large. Supercomputers based on vector or parallel architecture are generally used for this purpose.

PARALLEL ALGORITHMS

The most important part of parallel programming is to map out a problem on a multiprocessor environment. The problem must be broken down into a set of tasks that can be solved concurrently. The choice of an approach to the problem decomposition depends on the computational scheme. For the MacCormack scheme, one can observe that the calculation of the wavefield at a grid point at an advanced time level involves the knowledge of the wavefield at nine grid points of the current time level. Therefore, if we use a domain decomposition scheme for solving this problem second order neighbors will be involved in communication.

DOMAIN DECOMPOSITION

The parallel implementation of the algorithm is based on domain decomposition. Domain decomposition involves assigning subdomains of the computational domain to different processors and solving the equations for each subdomain concurrently. The problem domain is a cuboid and can be partitioned in three ways viz., stripe, hybrid stripe and checkerboard.

STRIPED PARTITIONING

In the striped partitioning of the 3D domain, the domain is divided into horizontal or vertical planes, and each processor is assigned one such plane. Striped partition can be done in three ways as shown in Fig.1.

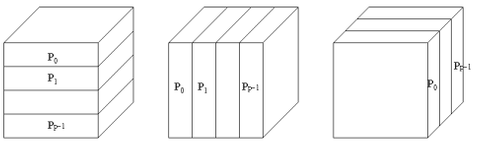

Figure 1. (a) Partition in z- direction, (b) partition in x- direction and (c) partition in y-direction

If we chose partitioning in z – direction then x-y planes have to be distributed among processors, if we chose partition in x- direction, then y-z planes have to be distributed among processors and if we chose partition in y direction x-z planes have to be distributed among processors. For load balancing we divide the domain in equal size of the pizza boxes, depending upon the number of available processors.

HYBRID STRIPE PARTITIONING

In hybrid stripe partitioning, partition is done using combination of two of the striped partitioning as shown in Fig.2.

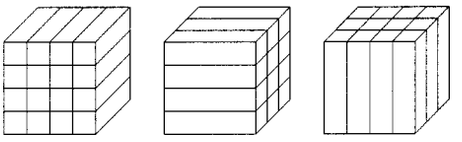

Figure 2. (a) Partitioning in z- and x- directions (b) partitioning in z- and y- directions (c) partitioning in x- and y- directions.

CHECKERBOARD PARTITIONING

In checkerboard partitioning, domain is divided in all three directions creating smaller subdomains. In uniform checkerboard partitioning, all sudomains are of the same size. These

subdomains have to be distributed among processors and no processor gets the complete plane (Fig3).

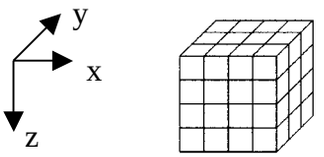

Figure 3. Checkerboard partitioning of the Domain.

INTERPROCESSOR COMMUNICATION

In the stripe partitioning as shown in Fig.1, each individual processor calculates the wavefield at each grid point in the corresponding subdomain at time k + 1, using wavefield values at previous time steps at the same grid point and its adjacent neighbors (depends on finite difference scheme used). The grid points can be updated using finite difference formula, simultaneously in all the subdomains except the grid points on the boundary that require information from the neighboring processors.

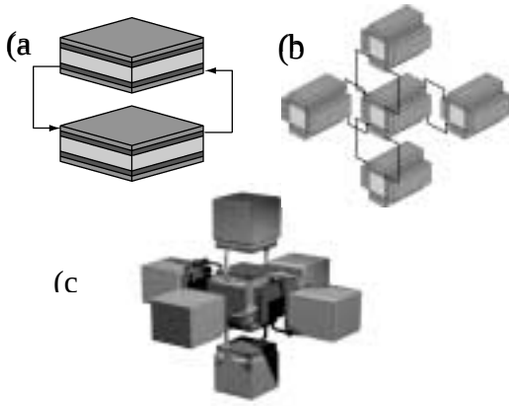

Figure 4. Communication between two adjacent tasks in (a) stripe, (b) Hybrid stripe and (c) checkerboard partitioning.

In order to calculate the wavefield at the grid points on the subdomain, at each time step, the required boundary grid points should be interchanged between the processors. Fig.4(a), shows the communication pattern in the stripe partitioning. For interchange of grid point, we attach an extra buffer layer (depth depends on the finite difference scheme) with the subdomains. The grid point(s) in the darker zone in the subdomain goes to the lighter zone (grey zone) of the other processor. Thus, in this case the two-way communication is in one direction only. This communication is known as ghost point communication.

In the case of hybrid stripe partitioning, ghost point communication is in two directions i.e. each processor should exchange boundary grid points with its four neighboring processors as shown in the Fig.4(b). While in the case of checkerboard partitioning the ghost point communication is in all three directions, i.e. each processor should exchange the boundary grid points with its six neighbors as shown in Fig.4(c).

ALGORITHM

In summary, the parallel algorithm for implementation looks as follows:

```
BEGIN
   Setup data structures, variables and constants
   Read velocity model
   Setup the domain decomposition
   Send the decomposed subdomain with corresponding
        velocity to different processors
FORALL processors simultaneously DO
  FOR every time step DO
     FOR every grid point DO
        - Evaluated wavefield
      END
   -Interchange border grid points with each adjacent
        layer(s).
   END
 - Gather wavefield from every processor
END
```

PARALLEL IMPLEMENTATION

Parallel implementations are being aggressively pursued on two fronts. The first is message-passing approach of cluster computing. The current leading technology for this is via Message Passing Interface, commonly know as MPI (Gropp, Lusk & skjellum 1999). The second philosophy is parallelism via shared memory loop level parallelism and available as a library called OpenMP (Chandra et al. 2000) for most of the shared memory (Symmetric Multi Processors) systems.

We have used MPI for the parallel implementation on a cluster of workstations. To setup the domain decomposition, we have used Cartesian topology approach of MPI. This helps in identifying the neighbors for the inter-processor ghost point exchange step for wavefield calculations. For ghost point exchange we have used MPI Send-Receive function. For all other parameters distributions usual MPI send, receive and broadcast functions have been used.

PERFORMANCE ANALYSIS

We have performed the benchmark tests of the parallel algorithm for a problem size 400 x 400 x 400 on PARAM 10000 system which a cluster of SUN E-450 workstations.

We have used three types of partitioning for the domain decomposition and have experimented with all the three types. For implementation point of view all three types of partitioning play an important role on the basis of memory access pattern and degree of concurrency. Theoretically, checkerboard partitioning has the best memory access pattern as the partitioned data can reside in the first level of the cache available. In the case

of stripe and hybrid stripe partitioning for access of data from memory may require swapping between first and second levels of cache, which is an expensive operation. Hybrid stripe partitioning has better access patter as compared to stripe partitioning. A bar chart of execution time verses number of processors for 3D acoustic wave modeling shown in Fig.5.

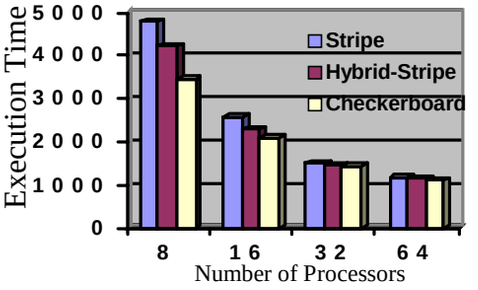

Figure 5. Bar chart for Stripe, Hybrid-Stripe and Checkerboard partitioning for 3-D acoustic wave modeling for model size 400x400x400.

When analyzing performance of a parallel algorithm we find that speedup is a function of problem size too. For a given problem size, as we increase the number of processors, communication to computation ratio increases. If we keep the number of processors constant, and increase the size of the problem, communication to computation ratio decreases.

SCALED SPEEDUP

The aim of the scaled speedup is to study the behavior of the parallel code when the amount of data per processor is kept constant while the number of processors is increased. In particular, this shows how the code would behave on huge problems when large numbers of processors are used (Gustafson, Montry & Benner 1988).

We have carried out experiments on two local problem sizes: (a) 64x64x64 grid points per processor and (b) 128x128x128 grid points per processor. In our experiment we have scaled the problem in all three directions. In this particular study, we define the scaled speedup as

$$S_p = \frac{p.T_2}{T_p}$$

where, T_2 is the elapsed time to solve problem on 2 processors and T_p is the elapsed time to solve on p processors. P is the number of processors.

DISCUSSION & CONCLUSIONS

The codes for these parallel implementations have been written using MPI message passing libraries. We have discussed a numerical example for showing the accuracy of the results. Performance analysis shows that for the domain decomposition, checkerboard partitioning gives the best performance as it has suitable memory access pattern for such problems. Checkerboard partitioning has advantage of achieving high degree of concurrency over other ways of partitioning. Table 1 (scaled speedup analysis), shows that we can achieve a good price performance ratio for smaller size problems on less number of processors and for large size of the problems we have to use large number of processors. Table 1, also predicts that very large size of the problem will scale well for very large number of processors.

Table 1: Scaled speedups for (a) 64x64x64 and (b) 128x128x128 grid point per processor

Number of Processors	**Scaled speedup** 64x64x64 grid points per processor	**Scaled speedup** 128x128x128 grid points per processor
2x2x1 = 4	3.59	3.9
2x2x2 = 8	6.61	7.6
4x2x2 = 16	12.95	15.2
4x4x2 = 32	18.13	24.6
4x4x4 = 64	24.45	35.8

ACKNOWLEDGEMENTS

Authors wish to thank C-DAC for providing computational facility on PARAM 10000 and permission to publish this work.

REFERENCES

Chandra, R., Dagum, L. Kohr, D. Maydan, D, McDonald, J. & Menon, R., 2000, Parallel Programming in OpenMP, Morgan Kaufmann Publishers.

Dai, N., Vafidis, A. & Kanasewich, E. R., 1996, Seismic migration and absorbing boundaries with a one way wave system for heterogeneous media, Geophys. Prosp., 44, 719-739.

Gropp, W., Lusk, E. & Skjellum, A., 1999, Using MPI - 2nd Edition, MIT press.

Gustafson, J., Montry, G. & Benner, R., 1988, Development of parallel methods for a 1024-processor hypercube, SIAM J. Scientific Computing, 9 (4), 609-638.

Mitchell, A. R. & Griffiths, D. F., 1981, The finite difference method in partial differential equations: John Wiley & Sons Inc.

Phadke, S., Bhardwaj, D. & Yerneni, S., 2000, Marine synthetic seismograms using elastic wave equation. Expanded Abstract, SEG 70th Annual International Meeting.

Sochacki, J., Kubichek, R., George, J., Fletcher, W. R. & Smithson, S., 1987, Absorbing boundary conditions and surface waves: Geophysics, 52, 60-71

Virieux, J.,1986. P-SV wave propagation in heterogeneous media: velocity stress finite difference method: Geophysics, 51, 889-90.

(Accepted 2002 November, 20. In original form 2002 August 27)