Parallel implementation of four depth-migration algorithms in a heterogeneous network

Baoniu Han

ABSTRACT

This paper covers the parallel implementation, using PVM (Parallel virtual machine) in a heterogeneous network environment, of four related depth-migration methods that are based on wavefield-extrapolation theory. Use of a master-slave topology, in which a common master code communicates with different slave processes, facilitates future development of new migration code since only the slave code needs to be changed to accommodate different algorithms. Automatic load balancing is incorporated in these codes to take full advantage of the available computing resources in a heterogeneous network. I show comparisons of efficiency tests on both shared and distributed computer architectures for prestack common-shot migration.

Key words: PVM, MPI, Wave equation, Depth migration.

Introduction

Migration algorithms that work in frequency-space and frequency-wavenumber domains parallelize naturally because frequency components are independent of each other, and hence can be processed separately. Here, I present parallel implementation of four different depth-migration methods: phase-shift-plus-interpolation (PSPI), split-step Fourier (SSF), implicit ω-x finite-difference (FD), and Fourier finite-difference (FFD). These implementations required little effort to convert from previous sequential migration code to the parallel code. Most of the initial converting work for all four methods was done within a weekend, though debugging of these codes took another few days.

The PVM library (Geist *et al.*, 1994), as opposed to MPI specification, is the communication library I used, the main reason for this choice being that the PVM library has more flexibility and power for computing within a heterogeneous network than does MPI, although some efficiency is sacrificed to achieve such characteristics. Murillo and Uzcategui (1997) suggested a general strategy on the parallelism of migration algorithms, and the PVM implementation here benefited from suggestions in Gouveia (1996) and Murillo (1996).

The motivation of this research is the need to compare the relative performance of different algorithms for migration applied to large volumes of 2-D poststack and prestack data. This parallel implementation turned out to be highly rewarding in that the complex Marmousi data, for example, can be migrated within half an hour compared to the nine hours required by the sequential codes. Benchmark tests performed on both shared and distributed computer architectures showed an almost linear relationship between the number of processors and speedup factors.

Though four algorithms are now parallelized, a pair of master-slave codes developed to accommodate the different algorithms into a common structure allows the ready generation of new migration code; only the downward-propagation part needs to be modified, facilitating the coding work for other algorithms.

Heterogeneous computing environment

The Center for Wave Phenomena (CWP) at Colorado School of Mines has about thirty workstations connected via an Ethernet network, forming the following heterogeneous computing environment.

1. Different hardware architectures

Among the thirty computers, 15 are PC workstations with Intel Pentium CPUs, their clock frequency ranging

from 90 Hz to 120 Hz. Nine of them are PC workstations with Intel Pentium Pro CPUs having a clock frequency of 200 Hz, and we have a six-processor SGI Power Challenge; the clock frequency of the MIPS-R8000 processor is at 75 Hz.

2. Different software environment

All the PC workstations were installed with the Linux operating system using free GNU development tools from FSF (Free Software Foundation). Although most of them have the latest Linux operating system, Slakware 3.4 with 2.0.30 kernel, some of them are still working with Slakware 3.2 (2.0.27 kernel) or 3.1 (2.0.0 kernel). The reason that we kept some of the old operating system is for the compatibility with older software that can run on only the old Linux system. As to the SGI Power Challenge, its operating system is IRIX 6.1.

3. Different network bandwidth

Most of the workstations are connected via the slow Ethernet network, with peak bandwidth at 10 Mbyte/s. Eight workstations are connected with a fast Ethernet network; it has a peak bandwidth of 100 Mbyte/s. When I test my implementation, I usually run a parallel job using 20 workstations, so my only choice is to cope with the 10 Mbyte/s slow network, which becomes a bottleneck for communication-demanding jobs.

Implementation considerations

Below is some of the key issues I have considered in the implementation of these migration codes.

• PVM versus MPI

To convert the sequential code to parallel code, I needed a communication library to adopt the master-slave topology. Two good choices are PVM (parallel virtual machine) and MPI (message passing interface) specification. Both of them are freely available from the web site of Netlib (http://www.netlib.org), with good documentation and some sample code. In my implementation, I chose PVM as the communication library based on a fair comparison of the above two systems by Geist et al.(1996).

Though generally less efficient than the MPI specification, PVM was designed at the outset to work for a heterogeneous-network computing environment. It has better portability and inter-operability performance than any other existing communication library. Specifically, the code written in PVM can be compiled and run on any hardware architecture and operating system without changing the source code; PVM jobs on different hardware architecture can communicate and exchange data with one another; and PVM supports both C and F77. Also C and Fortran code can exchange data between them even though arrays in C and Fortran have opposite storage sequences; for example, one can code a master program in C and a slave program in Fortran without any ill-effect. As to the MPI specification, though the source code written in MPI can also be ported between different architectures without any modification, it lacks the inter-operability feature: jobs on different platforms cannot exchange data with one another, and codes in C and Fortran cannot communicate with each other. The high efficiency of MPI can be achieved only in a homogeneous, distributed environment, thus sacrificing the flexibility that is necessary for heterogeneous network computing.

Furthermore, PVM contains all the necessary requirements of process control. It can start tasks, keep track of the running processes and stop them. In contrast, in the widely available MPI-1.1 standard, no function has been defined to start a parallel task within a program. Although MPI-2 will accommodate some of the process-control specification, no public-domain MPI-2 implementation has been available to date.

Though all the source codes I have implemented are written in PVM, considering the similarity in the communication functions between PVM and MPI, if desired, they could be converted to utilize the MPI functions without any difficulty.

• A common platform for different algorithms

The four related depth-migration algorithms I have implemented in both poststack and prestack shotgather domains are PSPI (phase-shift-plus-interpolation), SSF (split-step Fourier), FD (implicit $\omega - x$ finite-difference), and FFD (Fourier finite-difference) methods. All of the algorithms were derived from the one-way acoustic wave equation based on downward-continuation theory, and work in the frequency-space domain. These algorithms are naturally parallelized; that is, all the frequency components are independent of one another, and hence can be processed separately. For details of these algorithms, refer to Han (1998), Clearbout (1985), Ristow and Rühl (1994), Stoffa et al. (1990), and Gazdag and Sguazzero (1984).

In my implementation, I have adapted a master-slave topology. The master program distributes the frequency components to different slave tasks, and collects and stacks the final image after sending out all the data. Since the only differences among these migration algorithms are in the downward-continuation propagator, I code a common platform for all of them to make the code reusable, consequently saving implementation effort and facilitating future development of other migration algo-

rithms. The master code is identical for all the migration algorithms. In fact, at the same time one can run a master program to communicate to the slave tasks with different downward-continuation propagators. The communication part of the slave code is also the same among the different algorithms; only the core part — the propagator — differs. Changing a parallel code from algorithm A to B can be easily achieved by substituting the new propagator for the old one.

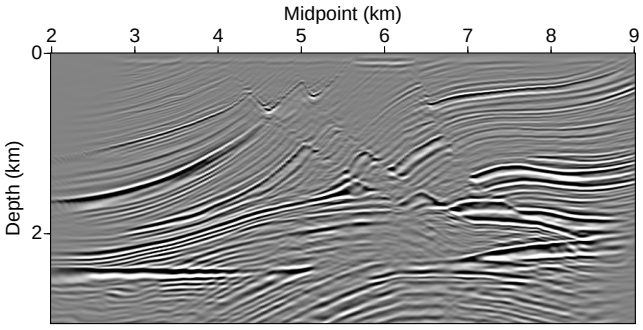

Figure 1. Marmousi section migrated with a parallelized phase-shift-plus-interpolation algorithm.

• Automatic load balancing

In migration, the total computational cost is determined by the size of the migration aperture, the frequency range of the input data, the grid spacing, the number of depth steps and the migration algorithm itself. Once the above parameters have been fixed, the total computing cost will be static and unchanged for a given hardware configuration. But as opposed to the static computing expense, the available computing resources in a heterogeneous computing environment can and do change dynamically. Besides the difference in the speed of the processor (for example, the Intel Pentium Pro 200 Hz processor is about twice as fast as the Pentium 90 Hz processor in floating-point computation.), the load of different workstations varies dynamically. Also, typically a parallel job needs to compete for the available resources with those of other users of the parallel network . In such situations, it is necessary to incorporate dynamic load balancing to take care of differences in computer speed and process load.

In parallelizing the four depth-migration algorithms, dynamic load balancing can be easily and readily put into the codes. Instead of dividing the entire data just by the available number of processors N, I multiply N by a data-dependent factor a (in my codes, it is usually set to 3 or 5), then partition the data using the new number aN. Smaller-size data sets are sent to different slave processors, and, once they are done, signals are sent back to the master process to request more data. In this way, with faster processors processing more data than slower ones, the data load between different workstations will be balanced. To reduce the network loading, all the intermediate migrated images are kept in the slave processors instead of sending them back to the master program. When all the frequency components have been processed, the master program will collect the partial images from the slave processors and stack them together to get the final image. Hence, the parallel implementation here will be generally computation, as opposed to communication, intensive, as is needed in a low-bandwidth network.

Benchmarking comparison

The performance measurements, done here on both shared and distributed computer structures, involve prestack migration on the Marmousi data (Versteeg & Grau, 1990).

Description of testing model and the measuring procedure.

The Marmousi data have been used extensively in the past decade to test the accuracy of prestack depth-migration algorithms and velocity analysis techniques on complex geological structures. It contains 240, 96-fold shot gathers, each trace with 726 samples and a sampling interval of 4 ms. The velocity profile used for migration is 375 by 369 samples, and the grid spacings are 25 m laterally and 8 m vertically. For the migrated Marmousi section shown in Figure 1, the migrated data were constructed with 60 frequency components uniformly incremented between 15 Hz and 35 Hz.

To test the performance of the parallel codes, instead of migrating the entire data set, I migrated only 15 shot gathers and saved the computing time for each shot gather to a file. After migration, the latter ten of the computing times were averaged to get a mean value. A speedup factor was computed by dividing the sequential time for a shot gather by the averaged parallel time. To test the relationship between the total computing time and the data-block size sent each time to the processors, I ran tests with, respectively, 1, 3, and 5 frequency components sent each time from the master process to the slave tasks.

The above tests have been repeated for all four algorithms with similar speedup results, showing that these implementations are independent of the algorithms

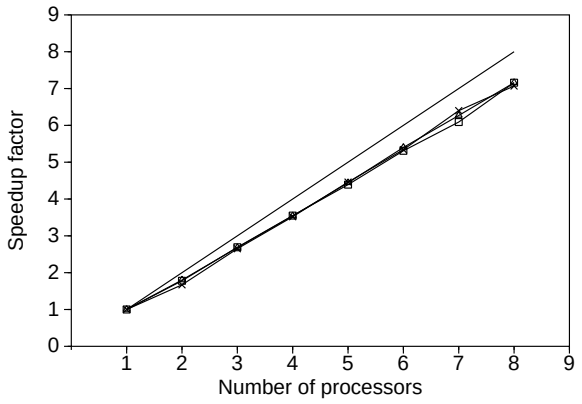

Figure 2. Speedup results for the 65^0 FD algorithm using an eight Pentium Pro based network; the straight line without marks is the ideal linear increase; the marked lines are for three different tests, in which different block-data sizes are sent each time from the master to slave tasks.

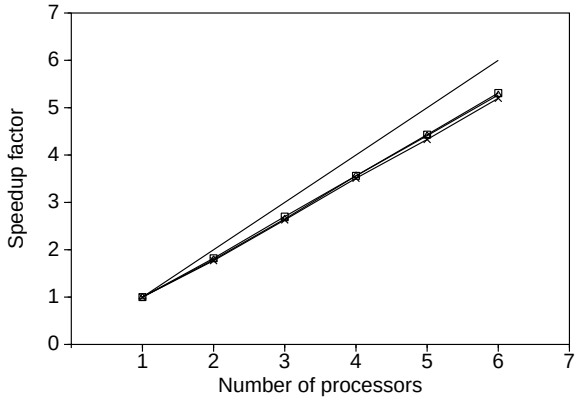

Figure 3. Speedup results for the 65^0 FD algorithm using a six-processor SGI Power Challenge; the straight line without marks is the ideal linear increase.

themselves. Below, is a discussion of the result for the 65^0 FD algorithm, as an example.

Performance comparison for the PC and SGI Power Challenge

To see what speedup we can get from these codes, the performance tests were done on two homogeneous platforms: a PC-based network with 8 Pentium Pro workstations and the 6-processor SGI Power Challenge.

The speedups achieved on the two platforms are shown in Figures 2 and 3. The speedup factors increase linearly on both testing platforms, although they still could not catch up with the ideal speedup factors. The difference between the ideal expectation and the test results is due to the time spent on communication through the network.

The communication time takes only a small portion of the total computing time, so the total computing time is almost inversely proportional to the number of processors, resulting in the somewhat linear increase of speedup factors in both figures. Though I tried different data sizes sent each time from the master to slaves, their results almost coincide with one another, again because these implementations are computation-intensive rather than communication-intensive.

In addition to the speedup tests here, I also performed load balancing tests for a heterogeneous network that include five different workstations: Pentium 90, Pentium 120, Pentium Pro 200, SGI Power Challenge, and SGI Indigo 2. For the codes without load balancing, each slave workstation works on the same amount of data. Using the SSF algorithm, for migration of a single shot gather of the Marmousi data, the code without loading balancing took about 68 as compared to the 39 seconds achieved by the code with load balancing. Without load balancing, more than 70% of additional time was wasted in waiting for the slower processors to finish their jobs.

Conclusions

The parallel codes appear to be stable (I ran PVM jobs on 20 workstations for 9 hours with no problem) and give satisfactory speedup factors. Using 60 frequency components (15 – 35 Hz) in a 16-PC heterogeneous network, the migration of the multi-offset Marmousi data took from half an hour for the 65^0 FD algorithm to two hours for the PSPI algorithm — the most expensive of the four approaches considered here. Later, when more PC workstations are connected via the fast Ethernet network, I will implement the MPI specification in these codes to see what, if any, improvement in efficiency is gained by using MPI.

Acknowledgments

Many thanks to John Stockwell for helpful discussions about SU(Seismic Unix). Also, thanks to Dr. John Scales who initially suggested that I do the parallel implementation.

References

Clearbout, J. F. 1985. *Imaging the Earth's interior.* Oxford, England: Blackwell Science Publishers.

Gazdag, J., & Sguazzero, P. 1984. Migration of seismic data by phase shift plus interpolation. *Geophysics*, **49**(2), 124–131.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., & Sunderam, V. 1994. *PVM: Parallel Virtual Machines, A users guide and tutorial for networked parallel computing.* Cambridge, Massachusetts: MIT Press.

Geist, G. A., Kohl, J. A., & Papadopoulos, P. M. 1996. PVM and MPI: a comparison of features. *http://www.netlib.org/pvm.*

Gouveia, W. 1996. SUDREF: a distributed elastic reflectivity modeling algorithm. *66th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts*, 986–989.

Han, B. 1998. A comparison of four depth-migration methods. *CWP Project Review (1998), Colorado School of Mines. Submitted to 68th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts.*

Murillo, A. E. 1996. DSU: Distributed parallel processing with Seismic Unix. *66th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts*, 997–1000.

Murillo, A. E., & Uzcategui, O. 1997. Three strategies for parallelizing a 3D seismic migration algorithm on distributed memory environments. *CWP Project Review (CWP-259), Colorado School of Mines*, 243–247.

Ristow, D., & Rühl, T. 1994. Fourier finite-difference migration. *Geophysics*, **59**(12), 1882–1893.

Stoffa, P. L., Fokkema, J. T., Freire, R. M., & Kessinger, W. P. 1990. Split-step Fourier migration. *Geophysics*, **55**, 410–421.

Versteeg, R., & Grau, G. 1990. The Marmousi Experience: Proceedings of the 1990 EAEG workshop on Practical Aspects of Seismic Data Inversion. *52nd EAEG Meeting. Eur. Asoc. Expl. Geophys.*