Efficient Parallel I/O for Seismic Imaging in a Distributed Computing Environment

Dheeraj Bhardwaj, Sudhakar Yerneni and Suhas Phadke*
Centre for Development of Advanced Computing, Pune University Campus, GaneshKhind, Pune 411007, India

SUMMARY

Parallel computers are evaluated by measuring their processor and communication speeds. But, for many large-scale applications the I/O performance is the bottleneck rather than the computational or communication performance. 3-D seismic imaging is one of such applications. Seismic data sets, consisting of recorded pressure waves, can be very large, some times more than a terabyte in size. It is always not possible to read and keep all the required data and information in computer memory. Therefore the data are partially read and intermediate results are written out. In this article we have discussed an approach to handle the massive I/O requirements of seismic migration and show the performance of parallel seismic migration code on PARAM 10000 which is distributed memory parallel computer. Use of parallel I/O clearly indicates the improvement of more than 30% in execution time.

INTRODUCTION

The processor and communication speeds of parallel computers have steadily increased, but the technology for improving the I/O sub systems has not progressed at the same pace. I/O subsystems for distributed memory parallel computers are often not designed to handle efficiently application with massive I/O requirements, such as seismic data processing. Most of the parallel computers work well with computationally intensive applications, but they are inefficient when it comes to satisfying the needs of applications that are also I/O intensive.

Recently, cluster of workstations or network of workstations has gained popularity as they provide a very cost-effective parallel-computing environment. Most of these clusters use Network File System (NFS) and MPI (Message Passing Interface) as message passing library. One limitation of NFS is that the I/O nodes are driven by standard UNIX read and write calls, which are blocking requests. This is not a problem for applications with small volume of I/O, but as the volume increases, it is necessary to be able to overlap computations with the I/O to maintain efficient operation (Olfield et al., 1998, Poole, 1994).

Although the I/O subsystems of parallel computers may be design for high performance, a large number of applications achieve only one tenth or less of the peak I/O bandwidth. The main reason for poor application-level I/O performance is that most parallel I/O systems are unable to handle efficiently small requests (Nieuwejaar and Kotz, 1996, Thakur et. Al, 1998), whereas parallel applications typically make many small I/O requests (on the order of kilobytes of less). This may be due to the overhead of an I/O call, but its most likely due to irregular access patterns and poor caching strategies.

One limitation of MPI–1 is that the I/O operation can be done only by master processor. The huge amount of data is read by the master from the input file & then distributed to the workers. As a result the workers remain idle during this period & minimizing the utilization of the resources. This also leads to the overheads of data distribution by the master to the workers.

ROMIO (A high-performance, Portable MPI-IO) is a very well suited software solution to a cluster environment where each machine has its own disk and processor. It provides a solution such that instead of a single processor reading the entire file and then scattering it to other processors, each processor does a local read or write.

The parallel I/O enables the processors to read in the required information by manipulating the offsets of the file. Hence the concept of master-worker no longer exists for distribution, thus speeding up the process of data input which directly affects the performance.

To image the underground geological structures three-dimensional seismic data sets are routinely migrated. This requires large computational power and I/O bandwidth

In this article, we present an efficient parallel implementation (with Parallel I/O) of a finite difference method based 3D post-stack depth migration algorithm on a distributed memory parallel computer.

3D POST-STACK DEPTH MIGRATION

Mathematical basis for migration in ω-x domain

Depth migration is necessary if the underground seismic velocities vary laterally and the wave equation based methods are common in use.

The migration method comprises of two steps, extrapolation and imaging. The extrapolation equation is a parabolic partial differential equation derived from the dispersion relation (Claerbout, 1985)

$$k_z = \frac{\omega}{v}\left(\sqrt{1-\left(\frac{vk_x}{\omega}\right)^2} + \sqrt{1-\left(\frac{vk_y}{\omega}\right)^2} - 1\right) \qquad (1)$$

where x, y and z are inline, crossline and depth axes respectively, k_x, k_y and k_z are the wavenumbers in x, y and z directions respectively, v is the velocity and ω is the frequency. By approximating the square root terms by continued fraction expansion, we obtain a 45-degree approximation. By inverse Fourier Transforming in x and z we obtain the parabolic partial differential equation. This equation is numerically solved by the method of splitting, which is the basis for the onepass approach (Phadke et al, 1997, 98). A Crank-Nikolson finite difference scheme with absorbing boundary conditions on the sides of the model is used for the solution. Imaging is the summation of all the frequencies at t=0 for each depth.

PARALLEL IMPLEMENTATION

The depth migration algorithm in ω-x domain is inherently parallel in terms of frequencies. The wavefield is first decomposed into monochromatic wave components by temporal Fourier Transform. The parabolic approximation of the wave equation in frequency-space domain is then used for downward propagate each monochromatic plane waves Therefore, each frequency harmonic can be extrapolated in depth independently on each processor and there is no need of intertask communication. One can introduce parallel task allocation into each frequency harmonic component with the ultimate goal being to have as many processors as frequencies. At each depth step all frequency components after extrapolation are summed up (Imaging Condition) to give the migrated image.

The stacked data is first Fourier transformed with respect to time and stored in frequency sequential format. Only the required number of frequencies are stored after Fourier transformation. The frequency bandwidth to be used for migration is determined from spectral analysis of the input traces. This forms the input data to the depth migration code. In addition to this a proper velocity depth model is also required.

Parallel Implementation with out parallel I/O

The parallel implementation is analogous to Master-Worker system. After reading all the required parameters, the Master determines the number of frequencies and frequency bandwidth to be assigned to each Worker. Then it reads and sends the frequency data to the designated Worker. Then the migration algorithm runs through the depth steps. The required velocity data for that depth step is sent to the Workers. Also the migrated data from all the Workers for that depth is collected by master, imaged and stored on the disk. A flow chart of this algorithm is shown in figure 1. The figure only shows one Master task and one Worker task, but in reality there are many Worker tasks. All the Worker tasks communicate with Master task in an identical fashion as shown in the figure 1.

Parallel Implementation with parallel I/O

The parallel implementation is analogous to SPMD (Single Program Multiple Data) system. Processors read all the required parameters and read the frequency data that is to be migrated by the individual processor, in parallel. Then the migration algorithm runs through the depth steps. All processors read the required velocity data for that depth step in parallel. The processor with rank zero, collects the migrated data from all the processors for that depth, images it and the stores it on the disk. A flow chart of this algorithm is shown in figure 2.

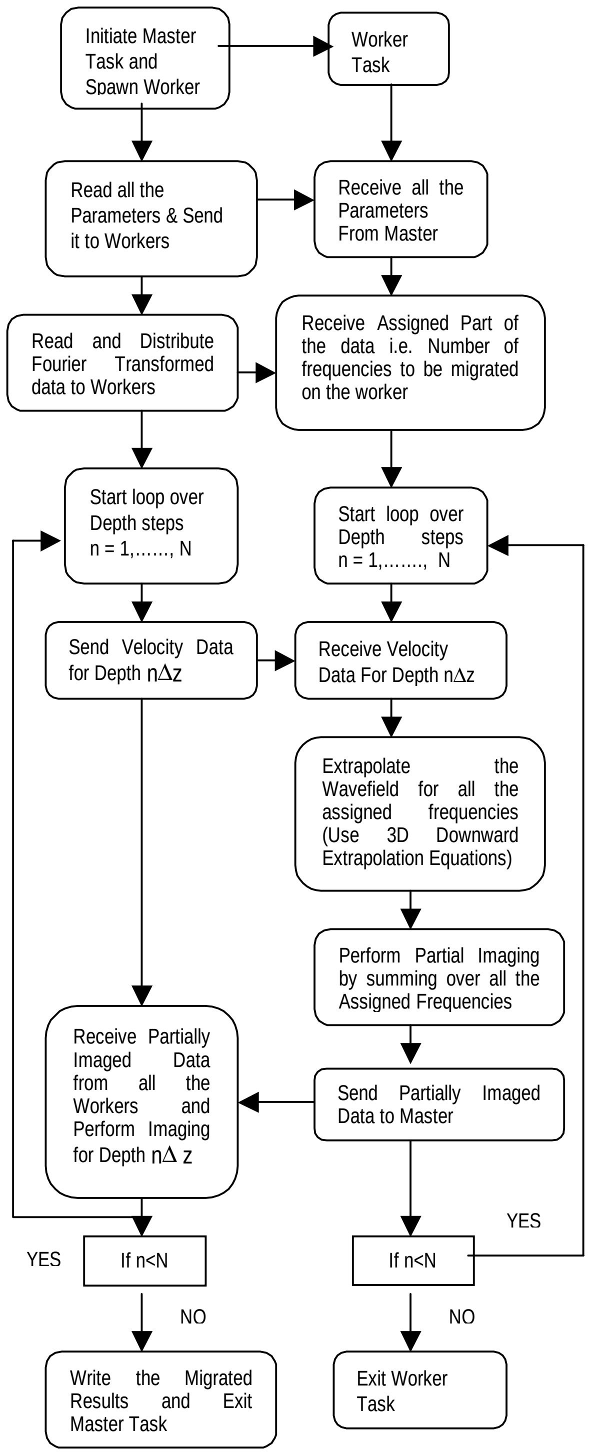
Initiate Master Task and Spawn Worker
Worker Task
Read all the Parameters & Send it to Workers
Receive all the Parameters From Master
Read and Distribute Fourier Transformed data to Workers
Receive Assigned Part of the data i.e. Number of frequencies to be migrated on the worker
Start loop over Depth steps n = 1,……, N
Start loop over Depth steps n = 1,……, N
Send Velocity Data for Depth nΔz
Receive Velocity Data For Depth nΔz
Extrapolate the Wavefield for all the assigned frequencies (Use 3D Downward Extrapolation Equations)
Perform Partial Imaging by summing over all the Assigned Frequencies
Receive Partially Imaged Data from all the Workers and Perform Imaging for Depth nΔ z
Send Partially Imaged Data to Master
YES
If n<N
NO
If n<N
YES
NO
Write the Migrated Results and Exit Master Task
Exit Worker Task

Figure1: Flow diagram of ω-x depth migration parallel algorithm without using parallel I/O

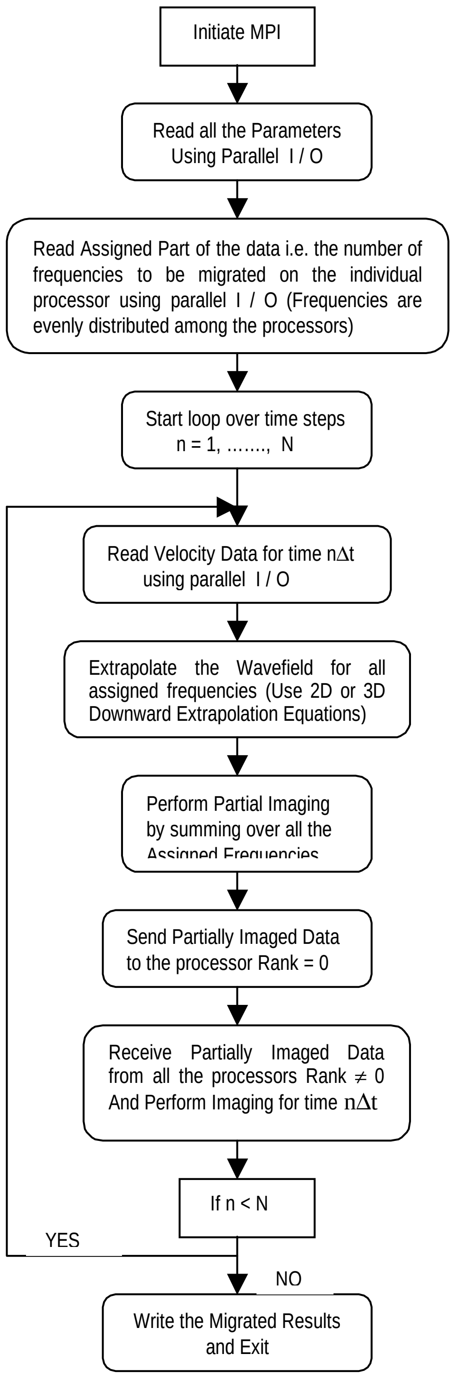
Initiate MPI
Read all the Parameters Using Parallel I / O
Read Assigned Part of the data i.e. the number of frequencies to be migrated on the individual processor using parallel I / O (Frequencies are evenly distributed among the processors)
Start loop over time steps n = 1, ……., N
Read Velocity Data for time nΔt using parallel I / O
Extrapolate the Wavefield for all assigned frequencies (Use 2D or 3D Downward Extrapolation Equations)
Perform Partial Imaging by summing over all the Assigned Frequencies
Send Partially Imaged Data to the processor Rank = 0
Receive Partially Imaged Data from all the processors Rank ≠ 0 And Perform Imaging for time nΔt
If n < N
YES
NO
Write the Migrated Results and Exit

Figure2: Flow diagram of w-x depth migration parallel algorithm using parallel I/O

PERFORMANCE ANALYSIS

Even though the developed codes for ω-x depth migration (with and without parallel I/O) are portable across various platforms, most of the development was done on PARAM 10000. The PARAM 10000 system has 40 SUN E450 compute nodes, each with 4 processors @300MHz. Out of 40 nodes 4 nodes are network file servers with 1GB RAM and 512K cache. High-speed network such as fast Ethernet with peak bandwidth 100MB/s connects the nodes. Following tables show the comparison between execution time for ω-x depth migration algorithms with and without parallel I/O for various real data sets of small and large sizes.

2D ω-x depth migration

Data details:

Size of fft data	2.66 MB
Size of Velocity model	1.824 MB
Total number of frequencies to be migrated	222

Execution time chart:

Number of Processors	Fast Ethernet (Time)	
	Without Parallel I/O	With Parallel I/O
16	57.8 Sec	37.6 Sec

3-D ω-x depth migration

Data Details: (Data Set 1)

Size of fft data	60 MB
Size of Velocity model	75 MB
Total number of frequencies to be migrated	420

Execution time chart

Number of Processors	Fast Ethernet (Time)

	Without Parallel I/O	With Parallel I/O
24	5765 sec	4331 sec
64	2301 sec	1645 sec

3-D ω-x depth migration

Data Details: (Data Set 2)

Size of fft data	1.3 GB
Size of Velocity model	1.2 GB
Total number of frequencies to be migrated	256

Execution time chart

Number of Processors	Fast Ethernet (Time)	
	Without Parallel I/O	With Parallel I/O
64	43500 Sec	28370 Sec

Note: Timings shown in the execution time chart are the average time of five consecutive runs.

DISCUSSION AND CONCLUSIONS

The execution time charts show that the execution time for seismic migration code with parallel I/O takes more than 30% less time as compared to the codes without parallel I/O. The reason for this improvement is the change in communication that is possible with the support of parallel I/O. The two flow charts explain the fact clearly. In the case of parallel I/O, the communication involved in the reading and distributing the data among processors can be changed to just reading the data in parallel by the processors without any communication involved. The communication comes at the end of the algorithm, when the final imaging takes place.

In the seismic industry, where the amount of data that needs to be processed is often measured by the number of tapes, which amount to hundreds of gigabytes or even terabytes, the improvement of making efficient use of the I/O subsystem becomes increasingly apparent. A 10% to 20% improvement in runtime would amount to saving of millions of dollars of processing time. The above mentioned results are a step in that direction

ACKNOWLEDGEMENTS

Authors wish to express their gratitude to the Department of Science and Technology (DST), Government of India, for funding the seismic data processing project under DCS. Authors also wish to thank the Centre for Development of Advanced Computing (C-DAC), Pune for providing the computational facilities on PARAM 10000 and permission to publish this work.

REFERENCES

Claerbout, J. F., 1985, Imaging the Earth’s interior, Blackwell Scientific Publications

Phadke, S. & Bhardwaj, D., Depth extrapolation of seismsic wavefields using cubic spline approximation, SEG (Society for Exploration Geophysicists) 67th Annual International Meeting, Nov. 2-7, 1997, Dallas Texas, USA.

Phadke, S., Bhardwaj D. & Yerneni, S., Wave equation based migration and modelling algorithms on parallel computers, Proc. of SPG (Society of Petroleum Geophysicists) second conference (1998), pp. 55 - 59.

Phadke, S., Bhardwaj, D. and Yerneni, S., Development of Seismic Migration & modelling algorithms for imaging crustal structures, Department of Science & Technology (Govt. of India) project report, 1998.

Poole, J., Preliminary survey of I/O intensive applications, Technical Report CCSF-38, Scalable I/O initiative, Caltech Concurrent supercomputing facilities, California Institute of Technology, Pasadena, 1995.

Nieuwejaar, N. and Kotz, D., Low level interfaces for high level parallel I/O. In Input/output in parallel and distributed computer systems, edited by R. Jain, J. Werth and J. Browne. Boston:Kluwer Academic, (1996), pp. 205-223.

Ron A. Oldfield, David E. Womble and Curtis C. Ober, Efficient Parallel I/O in Seismic Imaging, The Int. J. of High Performance Computing Applications, Vol. 12, No. 3, Fall 1998, pp 333-344.

Thakur, R., Lusk, E., and Gropp, W., User Guide for ROMIO: A High Performance, Portable MPI-IO Implementation, TM No. 234, ANL, IL 60439(USA), 1998.