Parallelization of 2-D and 3-D depth migration

J. Amundsen, Norwegian University of Science and Technology; B. Arntsen, A. Sollid, Statoil Research Centre; A. Buland, and R. Sollie, IKU Petroleum Research*

Summary

We have ported serial codes for 2-D and 3-D seismic depth migration to parallel platforms. The porting was done using portable parallel primitives, such that the same code can be run on a number of parallel machines, workstation clusters, as well as serial machines. Only small modifications of the serial codes were necessary. Efficient parallel codes were obtained with moderate modification of the serial codes.

Introduction

Downward continuation of seismic wavefields is one of the most widespread techniques for post-stack and pre-stack seismic imaging. Almost all 3-D poststack and 2-D prestack depth migration techniques, routinely in use today, rely upon the method of downward continuation in which the recorded wavefield is lowered to everdeeper recording planes by wavefield extrapolation. After a temporal Fourier transform of the recorded data, the downward-continuation can be performed independently on constant frequency -lines/-slices. These computations can proceed in parallel with a minimum of communication. Hence, the scheme is well suited for parallel processing. The actual codes used in this project are described by Sollid and Arntsen (1994) and Mittet et al. (1994).

When the seismic industry is looking for a l000-fold speedup over current performance, parallel seismic processing is a very rational choice. The cost of parallel computers is decreasing and their performance as well as their stability is increasing rapidly. Also, by running many high-performance workstations in parallel, one may achieve speeds comparable to those of supercomputers, at higher performance-to-cost ratio.

Parallelization strategies

Parallelization of the depth migration codes were performed using a SPMD (Single Program - Multiple Data) approach. Within this approach one node has a particular responsibility to read input parameters, distribute them to the other nodes, synchronize the execution, and collect and handle results in the end. We have in this work implemented a parallelization over frequency, where each processor is assigned particular set of frequencies. The processors will access the corresponding data, perform the depth migration and send their individual contributions back to the node that collects the data and constructs the final image.

A common strategy is chosen for the parallelization of the 2-D and 3-D codes. With the 2-D code, frequencies are distributed in blocks between processors to optimize I/O operations. For the 3-D code, the frequencies to be migrated are distributed cyclically among the processors. in a round-robin. This is necessary because of load balancing considerations, since the number of floating point operations increase monotonically with frequency. Also, because of the larger amounts of I/O within the 3-D code, I/O optimization by blocking frequencies is not an issue.

The actual frequency migration step consumes 97-99% of the total CPU time, depending of the system used. This makes the strategy of parallelization over frequency very efficient at low to moderate aggregate I/O requirements. With larger (i.e. 3-D) data sets and/or larger numbers of processors, aggregate I/O requirements becomes a significant part of the total wall clock execution time. Hence, asynchronous I/O and parallel file systems become important issues, allowing to overlap I/O and computations, and to distribute the I/O operations onto several (striped) disk subsystems. Within the 2-D and 3-D codes, a small portable library for doing asynchronous I/O from C and Fortran has been developed. The library is ported to Intel Paragon, CRAY systems, the Parsytec GCpp, and workstation systems supporting POSIX asynchronous I/O (e.g. DEC, IBM and SGI).

With very large data sets, the ability to share frequency planes among processors becomes important to be able to store the entire plane into memory. The most efficient scheme of decomposing a plane in general is decomposition by 2-D subsquares. This minimizes the surface-to-volume ra-

3-D Code Also for the 3-D code, two test cases were generated. Specifications of the two test cases are given in Table 2. Case 1 was the impulse response test, computed on a grid measuring 256 x 256 x 256 grid points in the inline-, crossline- and depth-directions respectively. The computational effort of the problem scales, basically, linearly with the number of frequencies and linearly in each of the three spatial coordinates. The input to the test is an impulse, while the output is an image of a half sphere. Case 2 was a real data test computed on a grid measuring 440 x 460 x 320 gridpoints in the inline-, crossline- and depth-directions respectively. The input data consists of stacked seismic data from the North Sea, which was originally used for initial 3-D interpretation of a discovery.

3-D code	Case 1	Case 2
$N_x \times N_y$	256×256	440×460
N_z	256	320
Recording time [s]	4	8
Sampling time [ms]	4	4
Max frequency [Hz]	60	45
Frequencies	122	370

Table 2: Benchmarking specification for *the 9-D code*

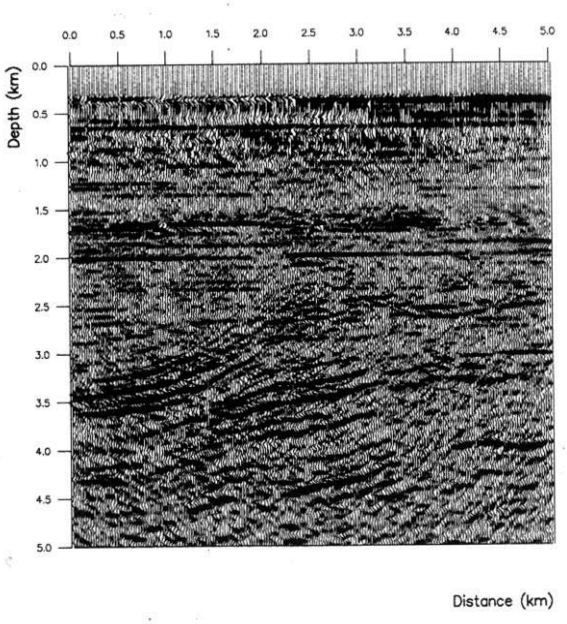

Fig. 2: A migrated section for Case 2 of the 3-D code.

Results and discussion

The results for Case 1 for the 2-D code is presented in Table 3. A program running on p processors with an execution time T_p will have a speedup

$$S_p = T_1/T_p$$

The speedup for the CRAY T3D scales very well with the number of processors, whereas the Parsytec scales poorly for this example. De spite a sequential speed, twice as fast as the Intel Paragon, at 32 nodes the execution time is about the same. On the other hand, the Paragon drops in speedup and efficiency when using more than 30 processors. The reason is that the performance is limited by I/O. The Paragon has 3 I/O nodes. When running on 56 nodes, each I/O node will serve about 19 processors. Ideally, this number should not be above 8-10 computational nodes per I/O node.

p	IBM Sp2	Parasytec GCpp	CRAY T3D	Intel Paragon
1	2311 s	6956 s	3632 s	11982 s
2	2.0	1.8	2.0	2.0
4	4.0	3.0	4.0	4.0
8	7.8	5.2	7.8	7.8
16		9.0	15.2	14.9
24		12.5		21.9
32		14.8		28.1
48				35.4
56				39.8

Table 3: Benchmarking results for Case 1 of the 2-D code

In Table 4 we. show the corresponding results for Case 2. The SP2 shows basically the same speedup for Case 1 and Case 2. However for the Parsytec we find that the speedup has improved significantly.

The results for the 3-D code are presented in Table 5. Note here that the execution time on few nodes was too long to be measured. Consequently, the speedup figures marked with a star are therefore in this case related to the execution time on four nodes as follows: $S_p^* = 4T_4/T_p$.

tio and hence the overall message traffic during the convolution step. However, for efficient parallel I/O we need large contiguous and preferably page aligned segments in memory. As a compromise, we have chosen sharing by partitioning the frequency plane into 1D slices among processors. This gives more message traffic within the convolution step, but avoids a separate collective buffering step with additional memory lost to buffering space before writing migrated images to disk. Notice that with the benchmark data sets and the parallel systems used, there has been no need for sharing frequency planes among processors.

Message passing

Message passing is still the most popular parallel programming paradigm even with the presence of the HPF standard. With the availability of the MPI standard in freely available source code and in optimized vendor libraries, portability of MP-programs has improved significantly.

However, to enable the use of more efficient low level vendor-specific MP-routines, a generalized communications package was developed. This interface mapped closely to the basic MP-calls of the MPI and PVM API's, In addition, it allowed for the use of vendor specific libraries by setting flags at compile time. Also, a sequential an parallel code was easily maintained without duplicating source, because all of the MP-code was left out at compile time by omitting any MP defines.

Benchmarking

The two platforms chosen for this project are the Parsytec GC/PowerPlus and IBM SP2. We have also run some tests on the Intel Paragon machine, and the Cray T3D.

The Parsytec GCpp to be used in this project is a MIMD computer with distributed memory, and 64 processors. Each node has a memory capacity of 64 Mb. If using more than 32 processors, one or more processor will have to share a 64 Mb node board, reducing the physically available processor memory to 32 Mb. The peak performance per node is 160 Mflops (64-bit). The maximum communication rate has empirically been measured to 3.2 Mb/s between neighboring processors.

The IBM SP2 was a MIMD computer with distributed memory, and 8 nodes were used. The nodes have a memory rate capacity of 2130 MB/s. The maximum communication rate has empirically been measured to 35 Mb/s, with the minimum network latency of 52 microseconds, between two arbitrary processors.

Description of test cases

2-D Code The benchmarking was performed on data from two test cases. Specifications of both test cases are given in Table 1. Synthetic data for Case 1 was modeled, corresponding to a 1.5 km streamer in a model of length 3.2 km and depth 2.56 km. The experiment consists of only 12 shots. Correspondingly, for Case2 we used data from a 2.5 km streamer in a model of length 6.0 km and depth 2.7 km. This is a model of reasonable size. The experiment consists of 120 shots. Each shot is, however, independent from the others, so the computing time will scale linearly with the number of shots.

2-D code	Case 1	Case 2
Shots	12	120
Receivers	151	251
N_x	320	600
N_z	256	270
Recording time [s]	2	4
Sampling time [ms]	4	4
Max frequency [Hz]	50	50
Frequencies	210	420

Table 1: Benchmarking specification for the 2-D code

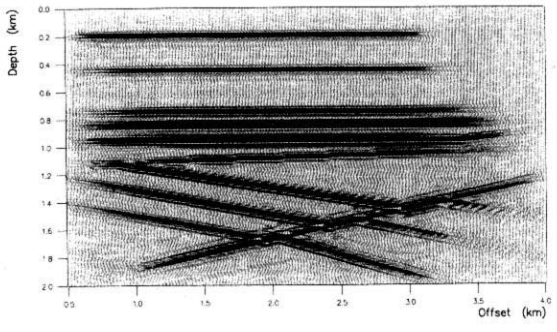

Fig. 1: A migrated section for Case 2 of the 2-D code.

p	IBM Sp2	Parasytec GCpp
1	83384s	192457s
2	2.0	
4	3.9	3.9
8	7.8	7.8
16		15.2
32		28.3

Table 4: Benchmarking results for Case 1 of the 2-D code

p	Case 1		Case 2
	IBM Sp2	Parasytec GCpp	IBM Sp2
1	48490s		
2	2.0		
4	3.9	81204s	82444s
6			5.6*
8	7.0	8.0*	6.4*
16		14.0*	

Table 5: Benchmarking results for the 3-D code

Conclusion

We have ported a serial code for 2-D and 3-D seismic depth migration to parallel platforms. The porting was done in a general way such that the same code can be run on a number of parallel and serial machines. The problem is well suited for parallelization, so only moderate modifications of the serial code had to be done. The results show that an efficient parallelization of the code was obtained. The execution time scales very well with the number of processors. We consider the results obtained for the parallelized program to be very encouraging.

Acknowledgment

We also acknowledge partial support from the CEC through the EUROPORT II project.

References

A. Buland, R. Sollie, and J. Amundsen. Parallelization of a code for seismic depth migration, volume 919 of Lecture Notes in Computer Science, pages 910-915. Springer Verlag, 1995.

R. Mittet, R. Sollie, and K. Hokstad. Prestack depth migration with compensation for absorption. In Expanded Abstracts, EAEG, 1994.

A. Sollid and B. Arntsen. Cost effective 3d one-pass depth migration. Geophysical Prospecting, 42:755-776, 1994.