Parallelizing a seismic inversion code using PVM: a poor man's supercomputer

Roelof Versteeg, Michel Kern and William W. Symes *

June 27, 1994

Abstract

This paper presents experience with parallelization using PVM of DSO, a seismic inversion code developed in The Rice Inversion Project. It focuses on one aspect: trying to run efficiently on a cluster of 4 workstations (in our case IBM RS6000, model 370). We use a coarse grain parallelism in which we dynamically distribute the shots over the available machines in the cluster. The modeling and migration part of our code is parallelized very effectively by this strategy; we have reached a overall performance of 104 Mflops using a configuration of one manager with 3 workers, a speedup of 2.4 versus the serial version, which according to Amdahl's law is optimal given the current design of our code. Further speedup is currently limited by the non parallelized part of our code (optimization, linear algebra and i/o).

1 Introduction

Differential Semblance Optimization ("DSO") is a variant of least-squares inversion of reflection seismograms, designed so as to overcome the non-convexity usually exhibited by least-squares formulation ([3], [9]). It has been described in detail in [6], [5], [8].

DSO is based on two modifications to simple least squares:

- first, one separates the different scales in reflectivity and velocity;
- the model is then *enlarged* to allow the reflectivity to depend on the position of the shot point. Since this does not make sense ("there is only one earth"), a penalty term is applied to impose that *neighboring* shots look alike.

Accordingly, the objective function is thus the sum of two terms:

- a least-squares misfit term, to force fit to the data,

*Department of Computational and Applied Mathematics, Rice University, P.O Box 1892, Houston, Texas 77251-1892,

- a differential-semblance term, to force the reflectivity to be independent on the location of the source.

Analysis shows that if this function is minimized first over the reflectivity, the resulting cost function, which only depends on the velocity, is smooth and convex and thus can be minimized effectively by gradient-based methods.

The definition of the DSO cost function J already requires the solution of a quadratic minimization problem in infinite dimension. Hence, in practice we will only be able to produce an approximation to its solution.

We solve the outer optimization problem using non-linear conjugate gradients, and we also use an iterative method (in the examples discussed here, conjugate residuals) to solve the inner, quadratic, problem. Thus, we need procedures to compute the forward map (modeling operator) itself, its derivatives, and their adjoints (migration operator).

Details on the implementation can be found in [6]. We just mention here that the evaluation of the adjoint operator uses the now classical adjoint-state technique ([4]). It turns out that the computation of the velocity gradient uses an extension of this method. The upshot of these considerations is that the basic computational block of the method is the *repeated solution* of the wave equation.

2 Considerations for code parallelization

Several factors decide the potential speedup and usefulness of a parallel code design, One is Amdahl's law which ([2]) helps us understand the performance we can expect from any parallel code. It states that if only a fraction f of the operations in a program can be carried out in parallel, the maximum speedup on p processors is:

$$S = \frac{p}{f + p(1-f)}.$$

This is bounded by $1/1-f$, regardless of the number of processors.

For distributed memory machines which must physically send data between processors (as opposed to shared memory machines) the ratio of communication vs. computation (expressed in wallclock time) is another factor. If this ratio is high, parallelization may be counterproductive. A third consideration is load balancing: different machines will have different speeds and different constraints in terms of other users. Thus static distribution of tasks over a number of machines wll often yield unsatisfactory results.

3 PVM

Our aim was to parallelize our code for use on a cluster of workstations. For this purpose we made use of one of the several message passing software packages explicitly designed for this use, namely PVM (Parallel Virtual Machine). It is public domain software developed primarily at Oak Ridge National Labs and Emory University. It enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. It seems to be the *de facto* industry standard with (according to a recent HPC newsletter) an estimated 70 percent of all distributed applications using it.

4 Structure of the code and parallelization

The results described in this paper pertain to 2D, constant density, linearized acoustics. However, the DSO principle is applicable to almost any physical model of propagation. Accordingly, our code is divided into a generic and a model dependent part. Procedures for linear and non-linear optimization, Fourier transform, linear algebra and i/o are implemented in a model-independent fashion in the generic part. The model dependent part of the code is generally where all the flops are and this was the part we choose to parallelize. For the 2D constant density acoustic examples treated here, the modeling method was finite difference approximation of the wave equation, of order 2 in time and 4 in space. The adjoint calculations amount to reverse time two way finite difference migration of shot records. The details of the implementation of this code are described in [6] who discussed loop level parallelization of this same code on Cray vector supercomputers using Autotasking (tm).

For the distributed implementation we choose a manager-worker configuration in which the manager distributes the data and tasks to the workers. The difference between the serial and parallel form are shown in Figure 1. The parallel code is identical to the serial code up to the generic - model dependent interface. In the serial code, the reading of data is followed by calls to the modeling subroutine; in the parallel code the reading of data is followed by PVM calls that pass these data to the workers. A driver program is resident on the worker machine that receives the data and calls the modeling subroutine, and subsequently sends the results back to the manager. When the manager receives results from a workers he sends a new task, thus ensuring a balanced load among workers (for a large enough number of tasks). To ensure that the workers are never waiting for the manager the manager starts by sending two tasks to each worker, thus implementing a crude form of prefetching and eliminating latency. Note that this design left 98 percent of our code unchanged: only the interface between the generic and the model dependent part had to be rewritten.

5 Example

We take as our test example for the efficiency of the parallelization the so called "gas cloud model" discussed in [7]. The data for this model are 30 shotgathers, thus we expect that we have sufficient data to make parallelization advantageous. We exercise a complete run of DSO, including linear and non linear optimization, and thus can make relevant evaluations of the speedup. Note that "a complete run" means that in the model dependent part we perform the equivalent of 58 prestack depth migrations of the whole data set (using two way reverse time migration). We have at our disposal a dedicated network of four IBM PowerStations model 370 connected by a 10 Mb/s Ethernet switch. In the following, all times will refer to runs of this example on this equipment.

5.1 Serial performance

In serial mode, our test runs at 43 Mflops on one IBM (and takes 33600 seconds to execute). We found that for the serial code 87 percent of time was spent in the model dependent part. Knowing this, we can predict the maximal speedup of our code and the possible speedup of our code if we increased this number to either 95 or 99 percent (Figure 2). We see that if we use two machines we expect a maximum speedup of about 1.76, and with 3 machines about 2.38. Note that the ratio of time spent in the generic and model dependent part do not correspond to the ratio of flops spent in these parts: in the generic code a large part of the cost is due to the out-of-core design of the code which includes disk to disk operations for such actions as vector adds; thus relatively slow disk i/o dominates the cost of this part.

5.2 Parallel performance

Using a configuration of one manager and two workers our code took 19380 seconds; a speedup of 1.73 and a MFlop rate of 75. Using one manager and three workers, our code took 13980 seconds, a speedup of 2.4 and a Mflop rate of 104. Note that the second number seems too good to be true: we seem to have violated Amdahl's law here. Actually we didn't: our timing of the model dependent part showed that we got some superlinear speedup due to overlap of i/o (on the manager) and computation (on the workers). Thus, our parallelization has given optimal results: our workers were kept busy all the time; their loads were balanced, and the communication costs appear insignificant.

6 Performance increase and limitations - discussion

The curves in Figure 2 tell a sad story: to double our speedup we would have to add 8 machines. Even though the individual machines can run our code at 43 Mflops, using them as part of a parallel network with this code only makes sense for a small number of machines. Note that [1] describes speedups of 5 using 6 workstations on the parallelization of a seismic migration code and that [8] describe similar speedups when using our code for migration only. The difference between these results and ours lies in the cost of the generic (optimization) part of DSO The generic part mainly involves two operations: dot products and subtraction of modeled from "observed" data. In the current implementation such operations are disk-to-disk and the i/o time is a factor 9 higher than the computation time (thus, parallelizing the computation would not be of any use without parallelizing the i/o) . Also, for the optimization we perform a gradient calculation which involves some bottlenecks in which all results get combined to get one number (this is a general problems of all parallelization implementations of optimization codes).

In summary, for the parallel design, choice of simulator, problemsize and hardware described here communications, latency and load balancing do not seem to limit speedup. Instead, the main limiting factor is the serial computation performed on the manager. In order to speed up overall performance the computation on the manager must be improved or distributed.

7 Conclusion

We have presented an efficient low cost implementation of an inverse problem solver on a dedicated cluster of workstations. We have shown actual performance measurements, and arguments which lead us to believe that for the current code adding more than two additional workers would not result in a significant speedup. However, we have achieved a sustained speed of 104 Mflops, which enables us to perform a large number of experiments inhouse at low cost.

8 Acknowledgments

This work was partially supported by the National Science Foundation, the Office of Naval Research, the Texas Geophysical Parallel Computation Project, the Schlumberger Foundation, and The Rice Inversion Project. TRIP Sponsors for 1994 are Advance Geophysical, Amoco Production Co., Conoco Inc., Cray Research Inc., Exxon Production Research Co., Interactive Network Technologies, Mobil Research and Development Corp., and Texaco Inc.

References

[1] G.S. Almasi, T. McLuckie, J. Bell, A. Gordon, and D. Hale. Parallel distributed seismic migration. *Concurrency Practice nd Experience*, 5(2):105–131, April 1993.

[2] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing. *AFIPS Proc. of the SJCC*, 31:483–485, 1967.

[3] P. Kolb, F. Collino, and P. Lailly. Prestack inversion of a 1d medium. *Proceedings of IEEE*, 74:498–506, 1986.

[4] P. Lailly. The seismic inverse problem as a sequence of before-stack migrations. In J.B. Bednar et al., editors, *Conference on Inverse Scattering: Theory and Applications*, pages 206–220, Philadelphia, 1983. SIAM.

[5] W. W. Symes and J. J. Carazzone. Velocity inversion by differential semblance optimization. *Geophysics*, 56(5):654–663, 1991.

[6] W. W. Symes and M. Kern. Inversion of reflection seismograms by differential semblance analysis: Algorithm structure and synthetic examples. Technical Report TR92-03, Rice University, July 1992.

[7] William W. Symes. DSO velocity inversion: a gas cloud synthetic example. Technical report, The Rice Inversion Project, Rice University, 1993. Submitted to Geophysics.

[8] William W. Symes and Michel Kern. Velocity inversion by differential semblance optimization fro 2-D common source data. In *SEG 62nd Meeting*, pages 1210–1213, Tulsa, 1992.

[9] A. Tarantola. *Inverse Problem Theory.* Elsevier, 1987.

FIG 1: Comparison of code structure for serial (left) and distributed (right) code.

FIG 2: Amdahl's law depicting maximum potential speedup as a function of the number of machines for a non parallel part of 13 % (A), 5% (B) and 1% (C).