Parallelizing Large-Scale Geophysical Applications in Java

M. Karrenbach * M. Jacob ** § M. Philippsen **

Abstract

Within Geophysics, seismic methods are an essential tool for petroleum and gas exploration. They produce images of the earth's interior and let explorationists analyze the geological structure of the underground. The memory and time requirements of seismic calculations suggest parallel implementations. In Fortran, however, parallel program code often not only lacks maintainability and reusability, but can be error-prone and looses portability when parallelization is attempted. This study shows that the advantages of Java (portability, parallel language constructs, strict object-orientation) can be used to efficiently implement basic seismic methods with acceptable performance. The performance of parallel High Resolution Velocity Analysis and parallel Imaging implemented in JavaParty is compared to the performance of High-Performance Fortran on IBM SP/2. Thereby a slowdown factor between Java and High-Performance Fortran of only about 1.5-2 has been achieved.

1 The Problem

Seismic methods are an important tool for petroleum and gas exploration. They aim at producing images of the earth's interior and let analysts determine the geological structure of the underground. Seismic methods process data obtained by reflections of elastic waves from different structural elements within the earth. The data are examined according to different speeds which distinguish the specific underlying structures. The necessary operations demand large amounts of processor time and memory.

The implementations of such methods in procedural programming languages like FORTRAN that are largely used in petroleum industry have some disadvantages:

- Procedural programming languages usually lack in software reusability and therefore make it difficult to maintain the source code. They don't involve an intrinsic hierarchy of operators which is needed for a compound framework of seismic methods.
- Message Passing libraries are used to implement seismic methods on parallel distributed memory computers. With these libraries the programmer does not only have to deal with the seismic problem but in addition has to tackle the key problems of parallel programming: load balancing and locality.

The Java programming language [1, 2] alleviates some of these problems. Java is object-oriented and offers language elements to express thread-based parallelism and synchronization without additional system libraries. In combination with the integrated

*Geophysical Institute, Karlsruhe University, Germany
**Department of Computer Science, Karlsruhe University, Germany
§part of this study has been supported by a diploma thesis grant of the Prof. Dr.-Ing. Erich Müller-Stiftung, Germany

Remote Method Invocation (RMI) of Java, data exchange between processors is easy. This allows for parallel processing on the application level, i.e., in the Java environment itself, as [3] discusses for geophysics. There is no further need for Message Passing or Shared Memory concepts within the application. Additionally, this parallelism is hardware independent.

From the software engineering point of view, Java seems to be a perfect programming language for the parallel implementation of seismic methods. Unfortunately, Java has the reputation of being slow and RMI of being even slower. However, in this paper we show that Java can achieve acceptable performance for scientific computing.

For this study we pick two basic seismic data processing operators: a velocity analysis and an imaging method. Efforts are underway at the Stanford Exploration Project to produce a seismic operator framework (JEST [8]) in Java. Our implementation of a parallel geophysical operator is based on a preliminary version of JEST.

2 Geophysical Applications

Geophysical seismic data sets often exhibit a regular data topology, that can be described by n-dimensional hypercubes. We consider here a pre-stack seismic data set which has 3 dimensions: recording time t, recording distance x, and source distance s. All axes are sampled in a regular fashion.

For the velocity analysis as well as the imaging procedure, we chose an integral transformation method. Thus the computations entail discrete summations over trajectories or surfaces in the seismic data volume. During the high resolution velocity analysis summation for a set of trial velocities is performed to determine summed energy's maxima and thus to determine the appropriate velocity. During imaging summation for a fixed velocity over a diffraction surface or curve is carried out for the previously determined velocity field. The final value of the summed amplitude corresponds to the scattering strength of an object in the subsurface.

The architecture of the parallel version of both operators is a combination of remote and local components. The data distributor starts threads on the remote machines after distributing source data among the processors. It is important to assign each thread to a different machine to obtain a well-balanced load between the processors.

Both algorithms can be parallelized efficiently, as is described below. For the purpose of comparison, these methods were implemented both in Fortran and Java.

2.1 Velocity Analysis

The main problem for parallelizing high resolution Velocity Analysis is the structure of the summation scheme. For a hypothetical velocity v that is to be considered during the analyses, all values from a $t \cdot x$ data subset array might contribute to the computation of the maximum amplitude.

Therefore, when the computation is spread to the processors along the s axis, every processor computes a matrix of partial sums. Then in a completion phase, all parts of the results have to be summed before getting the final output of this operator. The completion phase prolongs the total runtime of the operator considerably.

The parallelization along the velocity- or time-axis avoids the complexity of the completion phase and is hence more appropriate. If every processor analyzes some of the hypothetical values of velocity, every processor can compute a segment of the resulting matrix without having to communicate with other processors. Thus, the resulting matrix is produced by merging distinct result matrices together.

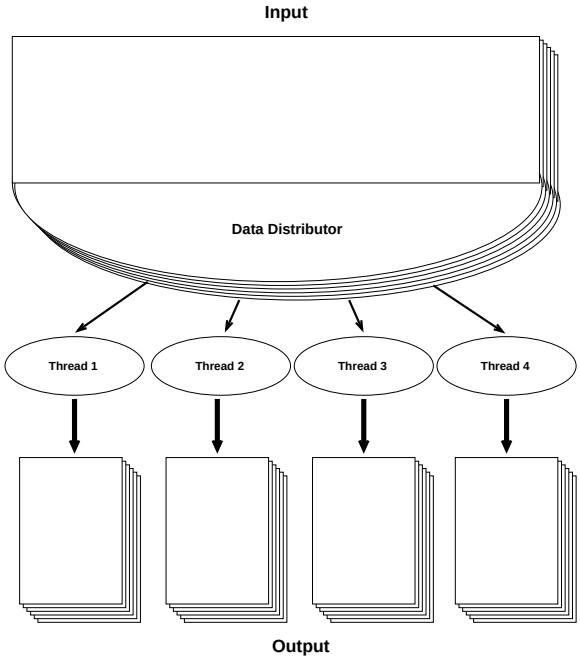

FIG. 1. *Architecture of the parallel implementation of the geophysical operators. The Data Distributor divides input data along the v-axis. Concurrent threads compute separate parts of the resulting arrays that can be merged together.*

For several input planes, i.e., for an additional source data plane, optimum performance is obtained if these planes are distributed among groups of processors with each group computing a distinct plane. Therefore, the resulting data space described by the t-, v- and s-axis can be divided in any way along the v- and s-axis, or the t- and s-axis, respectively.

2.2 Imaging

During imaging with integral transforms the $t-s$ data space is mapped into a $\tau-s$ output space, where the dimensionality and number of samples along each of the axes is maintained constant. The output space represents a picture of the scattering strength of the subsurface. Parallelizing this transformation proceeds by summing amplitudes along a diffraction curve for each of the surface positions s. Thus, all source positions can be treated independently in the output data space, however the input data space is partially common to many source positions.

Therefore, the source position parallelization strategy involves no communication overhead on output. Yet, the partial input data arrays need to be distributed on demand. We opted for keeping sections of the input locally in the processors in order to minimize communication between processors.

3 Implementation

The implementation of Velocity Analysis and Imaging in Java are conducted on an IBM SP/2. To ease the development and maintainance of parallel Java software, a precompiler called JavaParty has been used that is freely available [7, 6]. Software development with JavaParty is almost identical to coding in Java. If a class is marked by the class modifier `remote` adequate RMI hooks are inserted automatically.

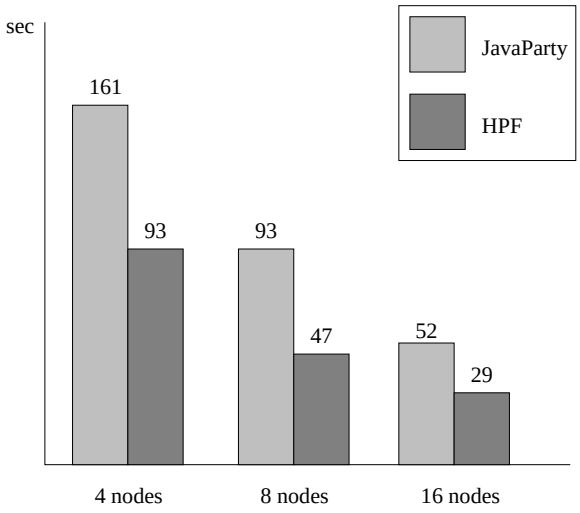

FIG. 2. *Measurements of parallel Velocity Analysis processing time on the IBM SP/2.*

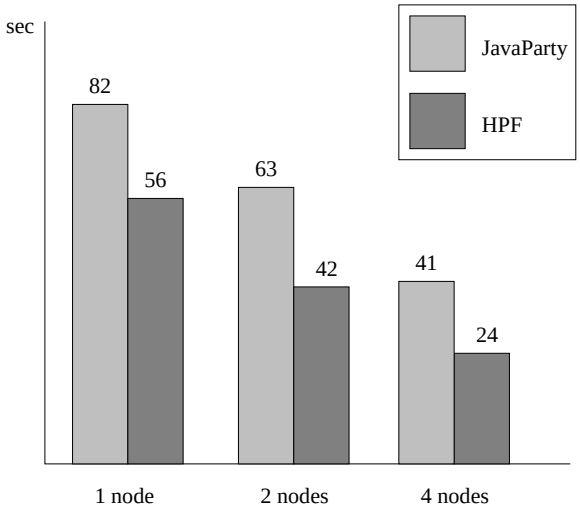

FIG. 3. *Measurements of parallel Imaging processing time on the IBM SP/2.*

In this section we compare the performance of parallel high resolution velocity analysis and imaging, which are implemented in JavaParty to the performance of HPF (High-Performance Fortran) versions and discuss the results. I/O has been excluded from the measurements since the emphasis of this paper is on parallel processing itself.

The performed benchmark on Velocity Analysis consists of an iteration with sixteen input planes with 1000 samples along the t-axis, 60 receivers, and 120 evaluated velocities. Figure 2 shows the results on four, eight, and sixteen nodes. The slowdown factor of JavaParty in comparison to HPF varies between 1.7 and 2.

The benchmark with Imaging conducts an iteration on a single input plane with 512 samples along the t-axis and 256 points along the s-axis. Figure 3 shows the results on one, two, and four processors. Here the slowdown factor of JavaParty in comparison to HPF ranges between 1.5 and 1.7.

4 Conclusion

We solved realistic seismic data processing problems in Java and observed a reasonable slowdown factor with Java. Furthermore, the measured results show that JavaParty adheres towards HPF speed. The slowdown factor is acceptable since JavaParty provides easy maintainability of source code, well-suited parallel language constructs, and it circumvents the need of Message Passing Libraries. Prospective implementations of Java compilers on SP/2 producing optimized native code as IBM's High Performance Java Compiler [4] will ameliorate performance issues further [4]. We achieved similar results with Velocity Analysis on SGI PowerChallenge. For more detailed information see [5].

5 Acknowledgments

We would like to thank the JavaParty group, especially Matthias Zenger, for their support of the JavaParty environment. Matthias Schwab of the Stanford Exploration Project designed and provided the geophysical operator framework JEST. Furthermore we want to express gratitude to Maui High Performance Computing Center as well as Karlsruhe Computing Center for granting access to the IBM SP/2.

References

[1] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[2] Siamak Hassanzadeh and Charles C. Mosher. Java: Object-Oriented programming for the cyber age. *The Leading Edge*, 15(12):1379–1381, December 1996.

[3] Siamak Hassanzadeh, Charles C. Mosher, and Calvin L. Joyner. Scalable parallel seismic processing. *The Leading Edge*, 15(12):1363–1366, December 1996.

[4] IBM. High performance compiler for Java. http://www.alphaWorks.ibm.com.

[5] Matthias Jacob, Michael Philippsen, and Martin Karrenbach. Large-Scale Parallel Geophysical Algorithms in Java: A Feasibility Study. In *ACM 1998 Workshop on Java for High-Performance Network Computing*, 1998.

[6] JavaParty. http://wwwipd.ira.uka.de/JavaParty.

[7] Michael Philippsen and Matthias Zenger. JavaParty - transparent remote objects in Java. *Concurrency: Practice and Experience*, 9(11):1125–1242, November 1997.

[8] Matthias Schwab and Joel Schroeder. Algebraic Java Classes for Numerical Optimization. In *ACM 1998 Workshop on Java for High-Performance Network Computing (Poster Presentation)*, 1998. http://sepwww.stanford.edu/sep/matt/jest.