Wave Inversion Technology, Report No. 1, pages 205-210

Large-Scale Parallel Geophysical Algorithms in Java: A Feasibility Study

M. Karrenbach and M. Jacob[1]

keywords: *java, velocity analysis, migration*

ABSTRACT

The memory and time requirements of seismic calculations suggest parallel implementations. In Fortran, however, parallel program code not only lacks maintainability and reusability, but is error-prone and loses portability when a message passing approach is used for parallelization. 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.

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. Seismic methods process data obtained from reflections of different structural elements within the earth. This is processed with a suite of different methods for obtaining relevant subsurface information. The necessary operations often demand vast amounts of processor time and memory.

The implementations of inversion methods in procedural programming languages (such as FORTRAN – largely used in petroleum industry) have 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.

  With the main focus on object-oriented languages the GOON/CLOP architecture (Clearbout and Biondi, 1996; Nichols et al., 1993) for implementing seismic methods in C++ has been developed within the Stanford Exploration Project. Although GOON/CLOP is appropriate for arranging operators for reusability, it suffers from performance problems, and it is not capable of being parallelized easily in a portable manner.

- Message Passing libraries are used to implement seismic methods on parallel distributed memory computers. With these libraries the programmer does not only

[1]**email:** martin.karrenbach@physik.uni-karlsruhe.de

have to deal with the seismic problem but in addition has to tackle the problems of parallel programming: load balancing and locality. The resulting program code is not only error-prone but cannot easily be ported from one machine architecture to another.

Recently, the Java programming language (Gosling et al., 1996; Hassanzadeh and Mosher, December 1996) has been introduced. Java is object-oriented and offers language elements to express thread-based parallelism and synchronization without additional system libraries. In combination with the integrated Remote Method Invocation (RMI) of Java, data exchange between processors is easy. This allows parallel processing on application level, i.e., in the Java environment itself, as (Hassanzadeh et al., December 1996) discusses for geophysical problems. There is no further need for Message Passing or Shared Memory concepts within the application. Additionally, this parallelism is hardware independent, i.e., programs run on any major platform.

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. But in this paper we show that Java can achieve acceptable performance for scientific computing.

For this study, we pick an operator for velocity analysis namely the Veltran operator which is a basic method in seismic processing. Efforts are underway at the Stanford Exploration Project to produce a seismic operator framework (JAG (Schwab and Schroeder, 1997)) in Java. We base our implementation of a parallel geophysical operator on a preliminary version of JAG. After a short introduction into the geophysical operator, the architecture of the underlying distributed runtime environment, JavaParty, is discussed, the parallel implementation of the operator is sketched. We compared the performance of parallel Velocity Analysis implemented in JavaParty to the performance of Fortran90 or HPF versions.

The benchmarks are conducted on up to sixteen nodes of an SGI PowerChallenge (shared memory parallel machine) as well as on the same number of nodes of an IBM SP/2 (distributed memory machine).[2]

The performed benchmark consists of an iteration of Veltran with a variable number of input planes with $n_t = 1000$ points along the t-axis, $n_x = 60$ points along the x-axis and $n_v = 120$ points along the v-axis. I/O is not considered during the measurements since each system platform has different capabilities. The measured time comprises the steps of data distribution, computation, and data collection. Measurements are evaluated within the computation of one, four, and sixteen planes. For the problem size of one plane, four nodes are sufficient, since no further speedup can be noticed with more nodes.

The JavaParty implementation on both machines is based on the JDK 1.1.2 with just-in-time compilers. On each node, a separate Java virtual machine is started. JavaParty implements the communication by means of RMI.

[2]The SGI had only sixteen processors. Therefore, we restricted the use of the IBM SP/2 to the same number of nodes to ease comparison.

On the shared memory SGI one could imagine a different approach where pure Java threads are used to implement the concurrency. Although desirable, the available JDK implementation currently does not support true parallelism of pure Java threads.

Although the IBM SP/2 offers an alpha version of a High Performance Java Compiler (IBM,) that compiles to native and statically linked code, it is too early to seriously use that compiler. The HPJ compiler revealed a general speed up of 1.6 compared to just-in-time Java performance, but unfortunately HPJ's communication through RMI turned out to be slower by a factor of between 20 and 35.

On the SGI, we used SGI's standard Fortran90 compiler with the switch `-pfakeep`. On the IBM SP/2, version 2.2 of the Portland Group High Performance Fortran compiler is used.

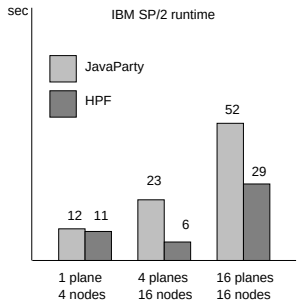

Figure 1: Measurements of parallel Veltran processing time on the IBM SP/2.

Figure 1 shows the runtime of JavaParty code versus HPF code on the IBM SP/2. The JavaParty version of parallel Veltran is slower than the HPF version by a factor of up to four depending on processor utilization. The factor of four can be noticed if too many nodes are used, i.e., if the per-node ratio of communication to computation time is not optimal.

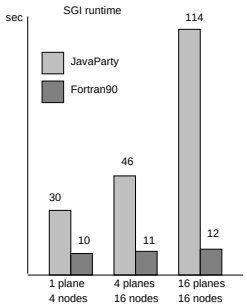

Figure 2: Measurements of parallel Veltran processing time on the SGI PowerChallenge.

Figure 2 shows the results for the same benchmark on the SGI PowerChallenge. Here the JavaParty implementation faces a slowdown compared to the Fortran90 implementation of three to almost ten. Since the SGI is a shared memory machine, the necessity to create separate Java virtual machines that communicate through RMI instead of using

pure Java threads in a common address space leads to severe disadvantages for the JavaParty implementation. However, we expect the factor ten to shrink significantly when truly parallel Java threads will become available in near future.

CONCLUSION

We achieved Fortran performance within a factor of between one and four with a parallel Java implementation of a basic geophysical algorithm on two major parallel platforms. Similar results can be expected for other seismic operators and on other platforms.

Java's performance is not attained by compiled and optimized native code but instead relies on interpreters with just-in-time compilation features.

We expect significant performance increases even on fine-grained computations in the near future because of two main reasons:

First, we had to use JDK 1.1.2, because later releases haven't been available for our hardware platforms. Later versions have increased performance, especially RMI performance, on Solaris and Wintel platforms and thus likely to show the same effect in our environment. Future releases (JDK 1.2) are announced to speed up the JVM even further.

Second, compilers producing native code like IBM's High Performance Compiler are likely to approach Fortran performance because they can apply much more sophisticated optimization techniques than current just-in-time compilers can afford.

ACKNOWLEDGEMENTS

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 JAG. Furthermore we want to express gratitude to Maui High Performance Computing Center as well as Karlsruhe Computing Center for the granting access on the IBM SP/2.

REFERENCES

Clearbout, J., and Biondi, B., October 1996, Geophysics in object-oriented numerics (GOON): Informal conference: Stanford Exploration Project Report No. 93.

Gosling, J., Joy, B., and Steele, G., 1996, The java language specification: Addison-Wesley.

Hassanzadeh, S., and Mosher, C. C., December 1996, Java: Object-Oriented programming for the cyber age: The Leading Edge, **15**, no. 12, 1379–1381.

Hassanzadeh, S., Mosher, C. C., and Joyner, C. L., December 1996, Scalable parallel seismic processing: The Leading Edge, **15**, no. 12, 1363–1366.

IBM, High performance compiler for Java:, http://www.alphaWorks.ibm.com.

Nichols, D., Urdaneta, H., Oh, H. I., Clearbout, J., Laane, L., and Karrenbach, M., August 1993, Programming geophysics in C++: Stanford Exploration Project Report No. 79.

Schwab, M., and Schroeder, J., May 1997, A seismic inversion library in Java: Stanford Exploration Project Report No. 94.

PUBLICATIONS

M. Jacob, 1997, Large-scale Parallel Geophysical Algorithms in Java — A Feasibility Study, interdisciplinary M.Sc. Thesis between the Geophysics and Computer Science Department at Karlsruhe University.

M. Jacob, M. Philippsen and M. Karrenbach, Large-scale parallel seismic Algorithms in Java, submitted to Leading Edge.

M. Karrenbach, Parallelizing large-scale geophysical applications in Java, To be presented at the SIAM mini symposium on Wave Propagation at the University of Colorado, Boulder, June 1998.