Wave Inversion Technology, Report No. 2, pages 323-329

Seismic Java Applications

M. Karrenbach and M. Jacob[1]

keywords: *Java, seismic, utility*

ABSTRACT

The ongoing computational project between the Geophysical Institute and Computer Science at Karlsruhe continued throughout the year. Parallel Java applications were tested on new platforms and with new releases of the Java Virtual Machine. Java numerical performance is still slower compared to Fortran. In addition, we designed new tools in Java for displaying the Stanford Exploration Projects plotting and animation file format.

PARALLEL SEISMIC PROCESSING IN JAVA

During the previous year we continued to evaluate the programming language Java as a potential candidate for writing seismic algorithms. As a standard benchmark we use the Mobil AVO data set and apply conventional prestack processing flows, including Velocity Analysis, Stacking and Kirchhoff Migration.

The programming language Java exhibits some attractive features for the computational scientists. These are, among others, object orientation and platform independence. However, when we are dealing with realistic sized seismic problems an ideal programming language should exhibit other important characteristics as well. Granted that portability of source and executable code is highly desirable, efficiency is of equal importance. Object orientation helps in developing, maintaining and extending software libraries and applications. Given the size of seismic processing problems, language constructs for parallelism should be included in such a language. In the end a multi processor application should scale well in a multi-processor environment.

We use the "JavaParty" package to hide explicit remote communication. It provides a mechanism to place objects on remote processors and start remote threads. Thus, the code we write, is Java Party code which is converted by the JavaParty compiler into standard Java code consisting of pure Java code and calls to Remote Method Invocations.

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

The following two code fragments compare the the expressions of parallelism in High Performance and Java for the example of a simple velocity analysis:

High Performance Code Fragment: Velocity Analysis

```
!hpf$ align data(*,*,i) with model(*,*,i)
!hpf$ distribute model (*,*,block)

!hpf$independent,new(is,s,ix,x,iz,z,tt,sx,it,wt)
do ip= 1, np {
do is= 1, ns { s = s0 + (is-1)*ds ; if(vel==1) s=1./s
do ix= 1, nx { x = x0 + (ix-1)*dx ; sx = abs( s * x )
do iz= 1, nz { z = t0 + (iz-1)*dz
       tt = z * z + sx * sx
 if(tt>0) { t = sqrt( tt ) }
 else     {t = 0 }
 it = 1.5 + (t-t0)/dt

 modl(iz,is,ip) = modl(iz,is,ip) + data(it,ix,ip) *sx

}}}}}
return; end
```

Java Code Fragment: Velocity Analysis

```
new DataDistributor(source, numThreads, EQUAL)
...
public void run() { float wt, sx, s, x, z, t; int it;
  dataF= new float[us-ls][nt]
  for   (int is= ls; is < us; is++) { s+=ds ; x= x0 + (float)nx*dx;
    for (int ix= nx-1; ix>=0; ix--) { x-=dx ; sx= Math.abs(s*x);
     z= t0 +(float)nt*dt;
     for(int iz=nt-1; iz>=0 ; iz--) { z-=dt;
       t= (float)math.sqrt((double)(z*z+sx*sx));
       it= (int)(0.5+ t-t0) /dt )
       if (it<nt)

         modelF[is][iz] +=(dataF[ix][it] *sx);
  }}}
  veltran.putDest(modelF,currentIndex, this.number);
}
```

These code fragments are representative for our benchmark examples which implement the integral transformations for velocity analysis

$$D(\tau, v) = \int_S f(seismic(t = m(\tau, h, v), h))dh \tag{1}$$

and for subsurface imaging

$$D(z, x) = \int_S f(seismic(t = n(z, x, x', v), x'))dx', \tag{2}$$

where $m(\tau, h, v)$ and $n(z, x, x', v)$ are trajectories of surfaces in the seismic data over which values or functionals are computed.

We performed the entire seismic processing sequence from Velocity Analysis to Stacking and Kirchhoff Imaging on the Mobil AVO data set.

BENCHMARK FINDINGS

The programming language Java achieves parallelism by using a threaded execution model and by using a Remote Method Invocation mechanism. Java was not designed for speedy floating point arithmetic calculations, yet seismic processing problems mostly use floating point arithmetic on large data sets.

We tested on a variety of parallel computers, including Sun SparcServer, SGI Origin2000, IBM SP-2 and clusters of workstations. The Java Virtual Machine was available on all parallel machine which we tested, however, the release number of the Java Developer Kit level was variable. Most of the machines provided a Just-In-Time compiler which speeded up execution considerably. In addition, on the IBM SP-2 a Java compiler which produces native optimized code was available and worked very well. For the Fortran comparisons we used the Portland Group High Performance Fortran compiler and the MIPS Power Fortran 90 compiler. We found that expressing parallelism via multi-threading constructs was convenient in Java and that Java is truly portable across all the tested machines on the byte code level. Thus recompilation is not necessary in a heterogeneous compute environment.

Java still exhibits the following deficiencies, which concern mainly the efficiency of code execution: The Remote Method Invocation is slow and there is no efficient support for Input/Output of floating point arrays. There is also no efficient support for complex numbers. Sophisticated numerical libraries written in pure Java are very rare.

Based on our benchmark we conclude that on most parallel machines the Java parallel byte code is up to 4 times slower than the equivalent parallel implementation in Fortran 90. However, if we forgo the advantages of having truly portable executable code by using a Java Native compiler, we achieve close to Fortran 90 performance. We

can obtain such a good speed up by compiling the Java language into native optimized code. On the IBM SP-2 we obtained a factor of 1.3 runtime difference for velocity analysis as well as Kirchhoff migration.

Results were published and presented at the SIAM meeting on Mathematical and Numerical Aspects of Wave Propagation in Golden, Colorado, 1998, as one of only two papers tackling the problem of parallel object oriented numerical computing in Java or C++. The Java Grande workshop plans to institute basic real-world computational benchmarks and we have been asked to contribute our source code for public distribution within that forum. Seismic processing continues to push the limits of numerical computational efforts.

JAVA DISPLAY TOOL

At WIT we continue to use the Stanford Exploration Project seismic processing package SEPLIB together with Colorado School of Mines' Seismic Unix system. The Stanford package produces device independent plot files that can be used on any computational platform.

We developed a display program in Java to view such plot and animation files. Thus we have the opportunity to distribute Java executables which will automatically run on any hardware platform and are able to present our research efficiently on the Web.

The tool allows to step through animated files and interprets the plot file on the fly. Right now this program is a stand-alone Java application. A web-centric Java applet version is planned to be available soon.

Figure 1 shows a screen snapshot of a single seismogram display done using the Java Utility. The functionality resembles the SEPlib program ”Tube”. Internally the utility contains a Java object that interprets the Vplot-specific plotting commands. The Java application relies only on the Java Virtual Machine and is thus platform independent. No recompilation is necessary. The executable byte code can be run anywhere.

CONCLUSION

We cooperate with interested research groups on parallel portable object oriented aspects of seismic computing. The programming language Java has many advantages compared to other programming languages, yet efficiency of floating point computations is still inferior to Fortran implementations. However with new Java native compiler the speed gap between Java and Fortran is closing, as we showed by using a realistic sized 2D conventional processing example. We develop tools that can be used

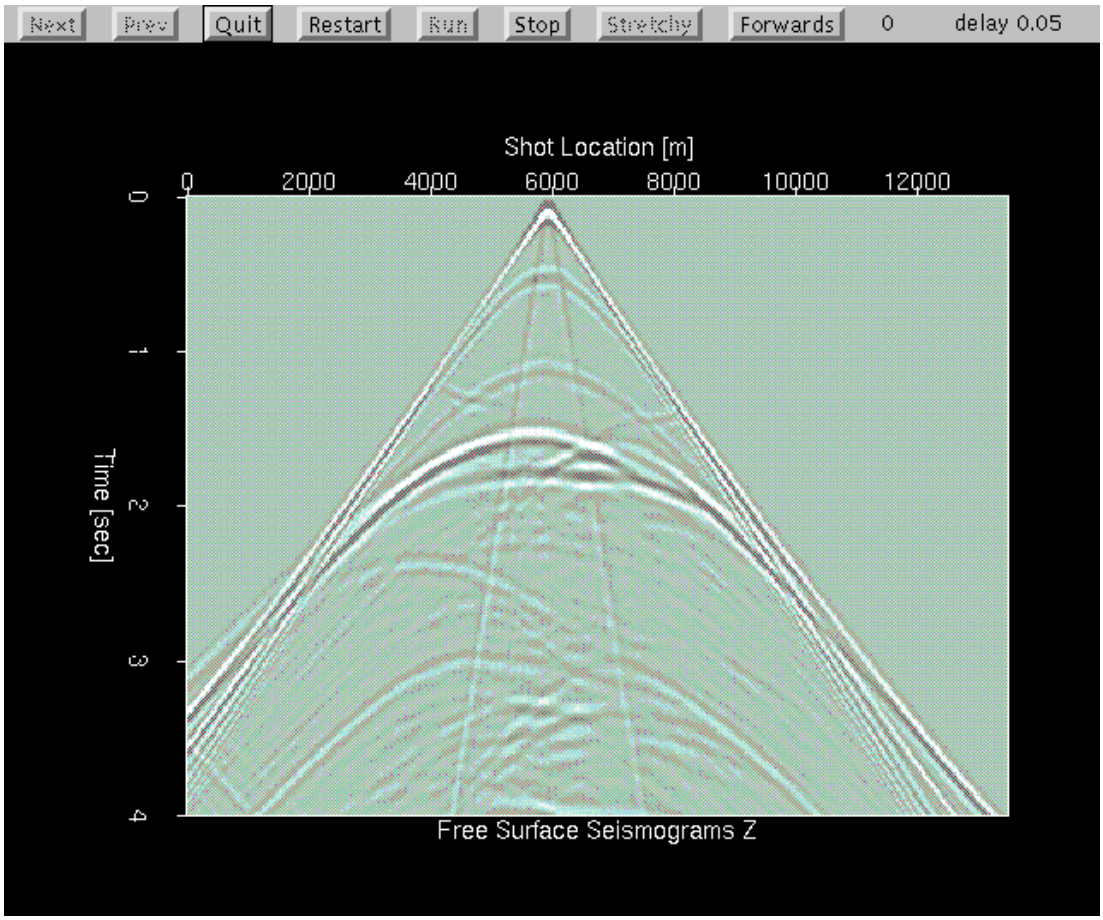

Figure 1: Screen snapshot of the display and animation utility written in the platform independent programming language Java.

"anywhere and anytime" (according to the Java philosophy).

ACKNOWLEDGMENTS

We appreciate support by M. Schwab, Stanford Exploration Project, HPCC Maui High-Performance Computing Center, Scientific Computer Center Karlsruhe, Silicon Graphics Inc. and IBM Research Labs.

REFERENCES

Jacob, M., Philippsen, M., and Karrenbach, M., 1998a, Large-scale parallel geophysical algorithms in java: A feasibility study: To appear in The Leading Edge.

Jacob, M., Philippsen, M., and Karrenbach, M., 1998b, Large-scale parallel geophysical algorithms in java: A feasibility study: Concurrency, Practice & Experience, **10**.

Jacob, M., Philippsen, M., and Karrenbach, M., 1998c, Large-scale parallel geophysical algorithms in java: A feasibility study: ACM 1998 Workshop on Java for High-Performance Network Computing, Palo Alto, California.

Jacob, M., 1997, Large-scale seismic processing in java – a feasibility study: Diploma Thesis.

Jacob, M., 1998, details online: http://wwwipd.ira.uka.de/ jacob/:.

M. Karrenbach, M. J., and Philippsen, M., 1998, Parallelizing large-scale geophysical algorithms in java – a feasibility study: Mathematical and Numerical Aspects of Wave Propagation, Society of Industrial and Applied Mathematics, 284–288.