Editor: Geoffrey Fox, gcf@indiana.edu

JAVA AND GRANDE APPLICATIONS

By Geoffrey Fox

SOON AFTER JAVA BECAME POPULAR FOR INTERNET APPLICATIONS, MANY OTHER FIELDS BEGAN LOOKING AT IT. WHILE JAVA'S ORIGINAL THRUST WAS CLIENT-SIDE APPLETS, TODAY ITS DOMINANT USE IS IN LARGE ENTERPRISE SERVERS RUNNING BUSness middleware. Its original client-side use is of secondary importance.

Java for high-performance computing has been a popular issue and the subject of annual workshops over the last seven years, including the latest event, Java Grande ISCOPE, JGI'02, (http://charm.cs.uiuc.edu/javagrandeIscope/) held in Seattle from 3–5 November 2002. This meeting series joined forces with the International Symposium on Computing in Object-oriented Parallel Environments (ISCOPE), aimed primarily at object-oriented methods in science and engineering with an emphasis on the use of C++.

Java Grande

These two conferences addressed OO methods in *Grande* applications. Grande describes a very broad class of problems that are large in some sense. These can include large massively parallel simulations as well as scalable approaches to large-scale distributed systems, areas discussed earlier in Grid and peer-to-peer computing Web Computing columns.

Several important research topics are included in the JGI (Java Grande-ISCOPE) agenda, which is divided into three broad categories: Java on the node, OO scientific computing, and distributed systems. Within these categories are

- Java for numerical computing with possible language and runtime issues
- High-performance Java compilation
- OO (largely C++ today) libraries
- Parallel Java and C++ environments
- Java-based distributed computing and Grid environments

Java on the Node

Initially, Java Grande introduced a severe performance penalty, up to two orders of magnitude slower than equivalent Fortran or C++ codes. Today, the penalty typically is no more than a factor of two on most scientific applications. To monitor performance, the National Institute of Standards and Technology developed SciMark (http://math.nist.gov/scimark2), a benchmark that averages several common scientific kernels including the fast Fourier transform and matrix algebra. The current top score is over 300 Mflops for the IBM Java VM (Virtual Machine) on an Intel architecture PC.

EPCC's Java benchmark suite (www.epcc.ed.ac.uk/overview/publications/research_pub/2001_pub/conference/jgflangcomp_final.ps.gz) compared Java with Fortran and C, with favorable results for Java reported at JGI'01. Much of this improvement came from better compiler techniques, including those reported at JGI meetings. Interestingly, Java uses dynamic just-in-time methods, and its compiler structure differs from other familiar languages.

The Java Grande community has discussed in detail (http://math.nist.gov/javanumerics) the key reasons why Java node performance cannot easily match Fortran and C++. These include

- Java floating-point rules
- Nature of Java arrays
- Overhead of *small objects*, as in complex data types
- Immature scientific libraries
- Lack of parameterizable types (which means each method must be replicated for each argument type: real, double, complex, and so on)

Substantial progress has been made since 1995 to give Java more flexibility in floating-point representations. Initially, the language insisted that Java should not only run on all computers but always give exactly the same answer.

Actually, this goal is quite interesting in applications in which exact reproducibility could be critical (medical instruments, for example). However, it is not natural for floating-point arithmetic in which different CPUs have different rounding characteristics even when operating with fundamentally the same IEEE representation.

The Java Grande community numerics working group relaxed Java's original strict rules with an optional `strictfp` modifier to enforce the original semantics. Java still cannot take advantage of several floating-point ac-

 Copublished by the IEEE CS and the AIP 1521-9615/03/$19.00 © 2003 IEEE

celerations—including the fused multiple-add instruction on some high-performance CPUs. However, the Java Grande community is no longer pursuing the addition of a `fastfp` modifier to enable this; perhaps current performance is good enough.

Java arrays are built as objects and create nontrivial overhead (they are not just a list of sequential entries in memory). Additional overhead occurs in using Java representations as objects for complex arithmetic or for, say, a three-dimensional velocity vector (or whatever physical structure describes the basic entities in the simulation). Here, the solution could involve language and compiler optimizations to remove the object overheads.

It is getting harder and harder to change Java because mainstream software systems use it so extensively, so most efforts focus on compiler strategies to recognize and remove overheads. While significant progress has been made, much of the advanced compiler technology remains only in research versions because major vendors see little demand from the scientific community for better performance. The Java Grande community is pursuing Fortran-style arrays in Java with JSR 83 (Java Specification Request) for such multi-arrays.

The next release of Java should include generic types, which partly address the parameterizable type issue mentioned earlier. Two other language enhancements of interest to scientific computing are *operator overloading* (so you could, for instance, add complex objects with the + notation and not with a non-intuitive add method) and *value classes* (objects represented by value and not by reference). These are not likely to appear soon. One interesting recent announcement from Visual Numerics is the availability of JMSL, a high-quality Java version (www.vni.com/products/imsl/jmsl.html) of its well-known IMSL scientific library.

All in all, Java performance on the node is reasonable today, but further improvement is possible in areas of importance to computational science and engineering. To hasten this requires users to show more interest and demand that new supercomputers offer Java compilers with the capabilities that have been proven in research but not yet deployed.

Parallel Java

Java for parallel computing involves both the node issues just discussed and parallelism support similar to that in other languages. There are several message-passing activities involved (such as Java bindings of MPI), shared-memory OpenMP, and other compiler-based approaches to parallelism. (For Indiana University work in this area, see www.hpjava.org.)

I will discuss these issues in subsequent Web Computing offerings but for now I can say that Java is as good or better than other languages in support for parallelism. Java has excellent communication libraries that are getting better with each release. Furthermore, the language is expressive and supports rich parallel constructs.

All in all, Java performance on the node is reasonable today, but further improvement is possible in areas of importance to computational science and engineering.

At JGI'02, researchers at Los Alamos National Laboratory and Rice University reported interesting results in developing large-scale scientific applications from scratch in Java and in using advanced compiler optimizations. They obtained good performance for the CartaBlanca code (www.lanl.gov/projects/CartaBlanca/overview.html) for heat transfer and multiphase flow and the Parsek particle in the cell code (charm.cs.uiuc.edu/javagrandeIscope/papersProgram.html#9). They also investigated different coding styles, from what we can call Fortran in curly brackets to a sophisticated style extensively using fine-grain objects.

Are these the approaches we should take? Should we give up some performance in exchange for a more powerful OO style and Java's robust software engineering? Is the additional complexity of C++ important? Perhaps Microsoft's C# is the answer? C++ and C# should perform better than Java, but so far, the scientific computing community has not extensively adopted them.

Distributed and Enterprise Java

Distributed systems have been the largest topic in Java Grande meetings because they build on Java's natural integration with the Internet. Much of the research in this area falls under the umbrella of Grid and peer-to-peer computing, which we have discussed in earlier columns and to which we will return in the future. This time I will touch on an area highlighted in the JGI'02 keynote talk by IBM's Pratap Pattnaik. He stressed the critical importance of robust enterprise systems that are today largely being built in Java.

Enterprise architectures are built around (Grid-like) architectures with individual components linked by messag-

How to Reach *CiSE*

Writers

For detailed information on submitting articles, write to cise @computer.org or visit http://computer.org/cise/edguide.htm.

Letters to the Editors

Send letters to

Jenny Ferrero, Contact Editor
jferrero@computer.org

Please provide an email address or daytime phone number with your letter.

On the Web

Access http://computer.org/cise or http://ojps.aip.org/cise for information about *CiSE*.

Subscription Change of Address (IEEE/CS)

Send change-of-address requests for magazine subscriptions to address.change@ ieee.org. Be sure to specify *CiSE*.

Subscription Change of Address (AIP)

Send general subscription and refund inquiries to subs@aip.org.

Subscribe

Visit http://ojps.aip.org/cise/subscrib.html or http://computer.org/subscribe.

Missing or Damaged Copies

If you are missing an issue or you received a damaged copy (IEEE/CS), contact membership@computer.org. For AIP subscribers, contact kgentilie@aip.org.

Reprints of Articles

For price information or to order reprints, send email to cise@computer.org or fax +1 714 821 4010.

Reprint Permission

To obtain permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at whagen@ieee.org.

ing subsystems. This architecture is replacing monolithic systems built around huge mainframes. Such approaches are perhaps inevitable with growth of distributed enterprises and commodity server systems. However, modularity and natural distributed support comes at a serious management cost.

Such enterprise solutions inevitably grow in size as Moore's law leads to smaller unit systems with increased performance coming from adding more CPUs. This is the *Grande Enterprise* (or Grid) problem. How can we manage and make robust such a continually growing decentralized system? This challenge requires perhaps new algorithms and software aimed at building what IBM calls autonomic systems (www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf).

Autonomic computers should sense their environments and respond properly to unexpected and often erroneous input. They should be tolerant of faults in themselves and others (a characteristic sought perhaps not so successfully through generations of human societies).

Scaling to ever more computers affects parallel and distributed computing differently. In the parallel case, you typically use SPMD (single program, multiple data) methodology with the same program on each node, so the key scaling problem is maintaining performance in terms of communication and load balance as the number of nodes scales up. In distributed computing, you typically scale heterogeneously, with each node running distinct codes; individual node or communication channel is important but not as critical as in the parallel case. Rather, the key scaling and performance problems stem from the system's heterogeneity and unpredictability.

As we solve the lower-level technical problems (better node and communication performance, for example), autonomic computing issues such as the robustness and management of Grande systems will become new foci of research. Here, we must refine and meld the different architecture styles of peer-to-peer and Grid systems to address these challenges.

Feedback?

Geoffrey Fox wants to hear from you. Comments? Criticism? Applause? Suggestions? What topic shall we tackle next? We want to hear about interesting topics and potential authors that could benefit Web Computing readers.

Don't hesitate to contact Geoffrey Fox at gcf@grids.ucs.indiana.edu.