ELSEVIER

Advances in Engineering Software 31 (2000) 743–751

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

Panel session: future directions and challenges for Java implementations of numeric-intensive industrial applications

M. Ginsberg[a,*], J. Hauser[b], J.E. Moreira[c], R. Morgan[d], J.C. Parsons[e], T.J. Wielenga[f]

[a]*HPC Research and Education, 35764 Congress Road, Farmington Hills, MI 48335-1222, USA*
[b]*Center of Logistics and Expert Systems, Karl-Scharfenberstr. 55-57, 38229 Salzgitter, Germany*
[c]*IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598-0218, USA*
[d]*Compaq Computer Corporation, MS ZKO2-3/N30, 110 Spitbrook Road, Nashua, NH 03062-2698, USA*
[e]*Engineering Systems International, 5330 Carroll Canyon Road, San Diego, CA 92121, USA*
[f]*Engineering Insight, LLC, 411 Huron View Boulevard, Suite 200, Ann Arbor, MI 48103, USA*

Abstract

This panel session focuses on utilization of Java for numeric-intensive applications, including the advantages and disadvantages of Java for future use with industrial independent software vendor (ISV)-based finite element methods (FEM) codes. Discussion will address both partial use of Java and/or complete code implementation; activities in this area are already in progress and have been reported by the Java Grande Forum (see http://www.javagrande.org). Some of the issues that will be discussed by the panelists and audience include: performance comparisons of Java, Fortran, C, and C++; primary deficiencies of Java with respect to future development of competitive commercial ISV-based FEM codes; Java standardization bottlenecks; strategies for transitioning to Java from existing large legacy commercial codes; current and future numeric-intensive benchmarks; actions to improve Java floating-point performance. This session is represented by biographical sketches of the panel participants, their individual reflections on the panel theme, and a list of related Internet references. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Java for numeric-intensive applications; Large-scale industrial simulations; High-performance computing; Parallel computation

1. Introduction

This session focuses on the use of Java to implement large-scale industrial simulations. The views expressed by the panelists are their own and do not necessarily represent the official viewpoints of their employers. The panel's collective experience reflects diverse interests in use of Java for numerical intensive applications, creation of ISV-based FEM software as well as development and utilization of optimized Java compilers and numerical libraries. In Section 2, all the participant surface and e-mail addresses are provided as well as phone and fax numbers so that anyone can follow up on any panel issues. Section 3 provides a brief biographical sketch of each person, while Section 4 presents individual viewpoints. Section 5 gives a few summary remarks and is followed by a list of references related to the panel theme and compiled by the participants.

Readers are encouraged to dialog with one or more of the panel participants to exchange viewpoints. Send suggestions for future panel themes to the panel organizer listed in the next section.

2. Participants

Myron Ginsberg, PhD, Organizer and Moderator
President
HPC Research and Education (HPC R&E)
Tel.: (248) 477-7018
fax: (248) 477-3129
e-mail: m.ginsberg@ieee.org

Jochem Hauser, PhD, panelist
Head, Parallel Computing Department
Center of Logistics and Expert Systems (CLE)
Tel.: 49-5341-875-401
fax: 49 5341-875-402
e-mail: jh@cle.de

Jose E. Moreira, PhD, panelist
Research Staff Member

* Corresponding author. Tel.: +1-248-477-7018; fax: +1-248-477-3129.
E-mail address: m.ginsberg@ieee.org (M. Ginsberg).

PII: S0965-9978(00)00018-1

Scalable Parallel Systems Department
IBM Thomas J. Watson Research Center
Tel.: (914)-945-3987
fax: (914)-945-4425
e-mail: jmoreira@us.ibm.com

Robert Morgan, panelist
Principal Member of Technical Staff
Compaq Computer Corporation
Tel.: (603)-884-0159
fax: (603)-884-0153
e-mail: bob.morgan@compaq.com

John C. Parsons, panelist
Development Manager
Engineering Systems International Corporation
Tel.: (619)-623-3992
fax: (619)-623-2799
e-mail: jparsons@cts.com

Thomas J. Wielenga, panelist
President
Engineering Insight, LLC
Tel.: (734)-913-8520
fax: (734)-913-8521
e-mail: tw@EngInsight.com

3. Biographic profiles

Myron Ginsberg, organizer and moderator, is currently President of HPC Research and Education. Dr Ginsberg has well over twenty-five years of high-performance computing experience in private industry (General Motors Research, EDS High-Performance Computing Group, HPC Research and Education), government research labs (US Army Research Laboratory, NASA Electronics Research Center, NASA Langley Research Center), and academia (University of Iowa, Southern Methodist University, University of Michigan). He has been significantly involved in General Motor's initial and continuing supercomputer efforts. He has edited four SAE volumes on automotive supercomputer applications. Myron has received the SAE Distinguished Speaker Award, the SAE Forest R. McFarland Award in recognition of his outstanding service in the automotive supercomputing field, and has been the recipient of the SAE Excellence in Oral Presentation Award. He has served as a distinguished national lecturer in high-performance computing for ACM, SIAM, IEEE, ASME, SAE, and Sigma Xi. He is also a Fellow of the ACM in recognition of his "pioneering and sustained contributions to supercomputing research and its application to the automotive industry". In his current position he has recently investigated for DaimlerChrysler the future use of Java for both large-scale numeric intensive computations and for automotive embedded systems. Myron has a BA and MA in Mathematics and a PhD in Computer Science (specializing in mathematical software and numerical analysis).

Jochem Hauser, panelist, is Head of the Parallel Computing Department at the Center of Logistics and Expert Systems (CLE) in Salzgitter, Germany. He is currently on sabbatical leave at ETH Zurich where he is teaching two courses in High Performance Computing with emphasis on object oriented programming (OOP) using Java for science and engineering applications. Since 1993, Dr Hauser has been a Professor of Computer Science and Parallel Computing at the University of Applied Sciences, Braunschweig–Wolfenbuettel, Germany and also Head of the parallel Computing Department at the Center of Logistics and Expert Systems (CLE) in Salzgitter, Germany. He is also a consultant to the European Space Agency in the field of aerodynamic simulation and high performance computing. Previously, from 1988–1992, Professor Hauser was the Head of the Aerothermodynamics Section at the European Space Research and Technology Centre (ESTEC), Noordwijk, The Netherlands of the European Space Agency. From 1985–1987 he was Professor of Computer Science at the Technical College of Landshut, Germany. From 1976–1984 he was a research scientist at a German National Research Center where he was leading a research group in environmental fluid dynamics, CFD, and numerical grid generation. Dr Hauser's research interests are in the development of algorithms for grid generation for complex geometeries as well as high-performance algorithms for parallel architectures in the area of computational fluid dynamics with specific areas of interest in numerical acceleration schemes and domain decomposition algorithms for structured multiblock domains for parallel processing. Dr Hauser's algorithm research efforts were honored with a NATO Scientific Affairs Divisional Award (1984) for environmental flow simulation and the NASA distinguished lectureship (1991). Dr Hauser previously taught at Hamburg University in the Department of Physics. He holds a Diploma in Physics from Giessen University in Germany (1973) and a Dr rer. nat. in space physics from Giessen University (1975).

Robert Morgan [34], panelist, is a Principal Member of Technical Staff for the Core Technology Group of Compaq Computer Corporation and a Senior Lecturer at Boston University's Metropolitan College Computer Science Department. He has been an active participant in computer research for thirty-five years of which the last twenty-five years has been involved in compiler development and language design. Currently, he is the designer for the implementation of a parallel language and leader of a research project on the efficient compilation of Java. Previously, he was a senior scientist at Compass, designing high-performance compilers for high-performance computers. Robert is a member of IFIPS Working Group 2.3 on System Implementation Languages and Compilers. He is also the author of the book entitled, Building an Optimizing Compiler, published by Digital Press.

Jose E. Moreira, panelist, is a Research Staff Member in

the Scalable Parallel Systems Department at IBM T.J. Watson Research Center. He received BS degrees in physics and electrical engineering in 1987, and an MS degree in electrical engineering in 1990, all from the University of Sao Paulo, Brazil. He received his PhD degree in electrical engineering from the University of Illinois at Urbana-Champaign in 1995. Since joining IBM in 1995, he has worked on various topics related to the design and execution of parallel applications. His current research activities include performance evaluation and optimization of Java programs and scheduling mechanisms for the ASCI Blue-Pacific project.

John C. Parsons, panelist, is Development Manager for Engineering Systems International Corporation. He came to ESI in July 1994 to head up a newly formed software development group. His team is responsible for the development, selection, and integration of software tools and components in their next generation applications. Prior to joining ESI, John spent 11 years in the CAD/CAM/CAE industry where he held positions in marketing, technical support, and research and development with General Electric, Calma, and Computervision.

Thomas J. Wielenga, panelist, is President of Engineering Insight, LLC, a company that specializes in the simulation of vehicle dynamics. He obtained his PhD (1984) in Mechanical Engineering and MS in Computer Aided Engineering from the University of Michigan. He received BS in Mechanical Engineering from Michigan State University.

Tom was involved in writing commercial dynamics software (ADAMS) at Mechanical Dynamics, Inc. (MDI). During that time he wrote and managed code having to do with sparse matrices, DAEs, and various modeling elements. He also taught courses on the theory of mechanical system simulation (ADAMS) for MDI and represented them in international competitions.

He started a company in 1989 to create a new mechanical system simulation program and in the process designed and implemented a modeling language (SimO) that was object oriented, and similar in many ways to the now pervasive Java language. Since then through the years he has followed the application of numerical algorithms and physical modeling in various languages including Fortran, C, C++, Java, and now Jpython.

Since 1979, he has been involved in modeling the motion of vehicles and machinery. He has modeled sport utility vehicles, chair-lifts, truck suspensions, and engines. He has designed and patented an engine and is patenting a device to prevent on-road rollovers in sport utility vehicles. He is a member of SAE, ASME, IEEE, and SIAM and is a registered P.E. in the State of Michigan.

4. Participant reflections

4.1. Myron Ginsberg viewpoint

From the perspective of twenty years in the automotive industry, I have to candidly admit that most implementations of large-scale, numeric-intensive automotive applications such as crash modeling and/or aerodynamic design simulations will very slowly adapt to total Java implementations. The worldwide automotive industry is dominated by the use of about a dozen commercial software codes primarily written in Fortran and C with a few in C++. The top priority with this class of automotive applications is speed of execution; this is extremely important as the simulation models grow increasingly complex while the lead time between vehicle concept and production continues to shrink with the major auto companies targeting lead times on the order of 18 months at present [16–18]. This means that the ISV-based codes must use very aggressive parallel implementations to reduce execution time per run. Most ISVs have fine-tuned their Fortran and C implementations to run very fast and until they can be convinced that Java implementations will run significantly faster than what they now have, they are very unlikely to switch, especially since they have large-investments in legacy code.

The argument that adoption of object-oriented code will drastically reduce code design and maintenance times will not be sufficient to sway these people because of the significant amount of legacy code, small staffs, and little or no time to do substantial re-writes. The ISVs who have already moved to C++ appreciate the reusability argument and are very happy with C++ with no current interest in switching to Java.

What is most likely to happen in the auto industry is a slow, conservative movement to Java usage. Already most ISVs appreciate the use of Java for graphical user interfaces (GUIs) as well as support for visualization and code segments written in different environments from the rest of their package. Moving to Java wrappers of legacy Fortran or C should be the next stage of evolution followed by use of Java fine-tuned numerical libraries [8,10,12] as they become widely available. The ISVs will still have to be convinced that the Java thread mechanisms to support parallelism provide at least the same performance levels as obtained on specific host machines using their native parallelism facilities.

The skeptical automotive ISVs will also have to be convinced that overall Java performance, especially speed, is superior to other existing approaches. The creation of some meaningful Java benchmarks that positively reflect on Java performance for attributes dominating industrial simulation problems would be very helpful to influence the ISVs. Furthermore, such benchmarks will have to explicitly demonstrate that Java approaches are indeed faster than obtainable in other languages such as C++. Certainly, the current work on JIT compilers and Java numerical libraries should help to impress the skeptic ISVs, IF progress in those areas rapidly continues. Also, the work at IBM Research clearly indicates that many of the current Java limitations can be overcome [26–33].

Another great concern about Java from the numerical analysis community is the behavior of Java floating-point

arithmetic. At present Java does not adhere to all the IEEE floating-point conventions and favors portability over accuracy on a given platform, i.e. opts for the same result across all platforms rather than the best attainable results on each platform. This has been discussed in several papers [19,25]. Sun has proposed some changes in this area [38].

Furthermore, if Java is to become the implementation language of choice, not only in the auto industry but also in significant segments of other industries, we will have to see grass roots industrial support for the necessary changes to make Java competitive in the numeric-intensive large-scale industrial simulation category. At present I don't see any evidence of such support. Indeed in trying to organize this panel I went to all of the automotive ISVs and only one agreed to serve. The others felt very reluctant to get up in public and say anything about Java. Privately, most were very negative and have more optimistic feelings about increased use of C or C++ rather than Java. Also, it should be noted that even with Fortran 90, 95, and 2000, object-oriented facilities can be used by those vendors who would be more content with object-oriented techniques in those environments with which they are already comfortable [20].

At present the visible supporters for Java for large-scale simulations seem to be mainly from academia as well as from a few government and industrial research labs [3,8,10,12,15,23,24]. If the Java Grande Forum [24] is to have any significant impact on private industry, it must convince those people of the benefits of Java implementations over other alternatives. At present, it is not obvious to me if that will readily happen in any industry. It will take some brave and innovative pioneers to lead the way and overcome the widespread skepticism. We have a few such pioneers on the current panel. The Symposium audience and readers of the proceedings should explore with an open mind the progress with Java illustrated in some of the bibliographic references below.

Perhaps identifiable groups from various industries need to be actively represented in the Java Grande Forum to express their specific needs for Java improvement for their applications.

Another problem is that there is too much HYPE associated with Java use and this along with disagreements about Java standards and the range of application can greatly impede progress. The potential market for use of Java is so large (particularly for areas like embedded systems), greed has motivated some vendors which can and are resulting in impediments to the creation of practical standards which, in turn, can essentially weaken the practical use of Java for some application areas.

Part of the conflict centers around portability vs. efficiency. For some applications such as large-scale, numeric intensive simulations, maximum speed and accuracy may require Java modifications and/or extensions or be practically impossible to implement. How much can Java be modified and still be Java and not fragment the original attributes of the language? Should there be some natural limits to the domain of Java applicability?

4.2. Jochem Hauser viewpoint

In the following, we briefly outline why we believe that *Java* should be and actually is *the* language for software engineering in science and engineering, and, in particular, for high-performance computing on parallel architectures in areas like computational fluid dynamics, computational physics, etc.

The release of the Java programming language by Sun Microsystems in late 1995 was an instant success with the Internet programming community. At that time, however, Java seemed to be unfit as a language for scientific and engineering programming because it was an interpreted language, and execution time was some two orders of magnitude higher when compared to corresponding C or Fortran codes.

Since then, Sun has released three major revisions, version 1.02 supporting distributed objects and database connectivity in 1996, and version 1.1 in 1997 added a robust event handling, Java Beans, and improved Remote Method Invocation (RMI). The most recent Java Development Kit (JDK), 1.2, appeared in late 1998, adding the Java Swing toolkit to produce portable graphics user interfaces (GUI).

When we compared Java with existing programming languages that are mainly used in science and engineering, namely Fortran and C (to some extent C++), despite its similar syntax to C, it became clear that Java was not just another programming language. Java is a fully object-oriented programming (OOP) language, providing, however, a much cleaner design than C++. OOP allows code construction reflecting, for instance, the engineering design process, because objects can be software coded and integrated. In addition, Java is the programming language for the Internet, and thus Java objects on disparate machines or even separate networks can be connected.

Producing engineering software in Java requires a different way of thinking; i.e. central to Java is the *class* concept. A class is a collection of data structures and methods, describing the functionality of a certain item, for example, a wing. An aircraft can be described by a set of classes, representing a wing, fuselage, nacelle, pylon, engine, etc. A specific aircraft can be constructed by instantiating objects from these classes. In this way, a direct mapping of the engineering parts to respective software objects can be achieved. The Java language mechanism allows *encapsulation* and *inheritance*, meaning that an existing class can be used and modified according to the needs of the code designer. The validated parent class will not be touched, allowing complete code reusability. The *interface* notion of Java extends the concept of inheritance, providing some kind of template. The Java OOP approach provides profoundly improved software productivity. Java's robust mechanism for *exception* handling promotes code

reliability, a feature considered to be essential for today's large and complex codes.

Most important for parallel computing, Java provides the *thread* concept. This is a lightweight process, allowing starting hundreds or even thousands of these threads from within a Java application and thus achieving concurrency. The mapping of threads to processors as well as thread scheduling is done by Java and the OS (virtual shared memory required; no distributed Java virtual machine exists at present). A major requirement to industry is to provide not only advanced thread schedulers, but also a standard API to allow the creation of custom schedulers. Threads also provide a way to obtain dynamic load balancing for a parallel application without explicitly assigning tasks to processors: a threaded application is said to be *self-scheduling*. Java also provides a mechanism for synchronizing threads and for sending messages between threads. No additional message passing libraries or language dialects are needed. Moreover, Java supports the *client–server* concept through remote objects, implemented by RMI. The engineer, wanting to perform a computation, starts the server (parallel machine) from his client (workstation) by providing proper authentication to the simulation code's security mechanism, that is part of Java. He then sends his own solver version at run time in the form of a remote object to the server code, i.e. replacing the default solver on the server. In this way, a multitude of different applications can be constructed, concentrating on the actual physics and mathematics, while the code infrastructure is already in place.

In all discussions we had so far, Java's lack of performance has been cited. This argument, at best, is of no concern now. First, just-in-time compilers (JIT) are now available, resulting in enormous speedups over the interpreted Java code. Using the latest *jitc* of JDK 1.1.6 from IBM alpha works for Linux Kernel 2.2.10, and writing a simple matrix multiplication code, $C = A \times B$, resulted in a floating-point performance of 24.8 MFlops on a Pentium II, 300 MHz processor (the JDK for Linux is in an early stage and provides no optimization). During my oral presentation I will give some floating-point numbers for comparisons of Java and C on the Linux OS. It should be noted that the IBM compiler is a pre-release alpha version that does not provide optimization at present (July 1999).

The design of any software should strictly follow Kernighan's rule *Make it right before you make it faster.* The key issue for large-scale parallel Java code is *parallel scalability*, i.e. will the Java thread concept deliver high parallel efficiency for large number of processors. The results of tests using the Caltech HP V class machine (32 processors, July 99) will be presented and discussed. The other important question addresses the quality of the *dynamic load balancing*. We will present speedup results resulting from improvement in compiler technology, in particular IBMs and Sun's Hotspot compilers. These numbers shall serve to illustrate the dynamic, only two-year-old history, of Java compiler technology. Second, speeds between 80 and 90% of corresponding Fortran programs were reported by a group of scientists (for instance see Ref. [3] of IBMs Watson Research Center. This group also provides Java numerics classes.

In the panel presentation the design philosophy and computational results of the *JParNNSS* [2], the Java parallel Navier–Stokes solver—an EC and Ministry of Science and Culture of Lower Saxony, Germany—funded program, will be reported. We will also present the Java parallel test suite for science and engineering to demonstrate the viability of the parallel thread concept for a range of engineering and scientific computations.

Java also harnesses the power of visual and graphics programming, providing the Swing toolkit for portable 2D graphics and GUI programming as well as a 3D application programmer's interface (API) for three-dimensional graphics, based on the Open GL standard.

The success of Java mainly stems from the explosive growth of the Java class library, providing features for numerical intensive computation, multimedia, database connectivity, native methods (calling of C code), network programming. This will allow the code designer to address all questions of computer simulation using a single language. By reusing the classes provided from various sources, Web distributed software-engineering projects can be tackled that were virtually impossible before the advent of Java.

There is a price to pay, however, because the legacy codes will have to be rewritten in Java. We feel that this task might not be as cumbersome as perhaps anticipated, because application specific routines can be directly converted into Java, while the structure of a general parallel simulation code for handling complex geometries only has to be written once. Only the actual solver for the specific engineering application would have to be provided. Again, we would like to cite another one of Kernighan's rules *Don't patch bad code, rewrite it.* There is always a compromise between flexibility and efficiency, and in the past, the Java language has been viewed as only flexible, but delivering reduced efficiency. The new generation of Java compilers, for instance by Sun and IBM, is beginning to change this by opening what we call "efficiency windows". Further speed increase is achieved through advanced thread management as demonstrated in HPs V class architecture.

Software is the lifeblood of our IT society, and a major effort has to be made to bring software development cost down and software reliability up. Otherwise, we might face the anachronism of running Fortran on the future quantum computer.

We conclude our remarks, by citing the *Java Grande Forum* (www.javagrande.org):

> Java has potential to be a better environment for 'Grande application development' than any previous languages such as Fortran and C++.

This statement is fully backed by our own programming experience of two decades as well as using numerous programming languages, and by writing Java codes over the last 18 months for large scale problems in science and engineering.

4.3. Jose Moreira viewpoint

For the past two years, the Numerically Intensive Java [3] team at IBM Watson has been developing techniques to improve the performance of Java in scientific and engineering applications. We have developed an optimizing Java compiler that typically produces between 80 and 100% of the performance of corresponding Fortran codes [7,31]. We have also developed linear algebra libraries, coded entirely in Java, that achieve 80% of the performance of the best existing native libraries.

We believe that these levels of performance, when combined with the software engineering advantages of Java, are enough to attract many applications developers. Nevertheless, we want to go a step further and demonstrate that Java can deliver better performance than Fortran for important classes of applications. In the next paragraphs I will briefly summarize the techniques we have developed to achieve high performance in numerically intensive Java code.

As identified by the Numerics Working Group of the Java Grande Forum [23,24], there are three major performance issues with respect to Java that have to be addressed: (1) multidimensional arrays; (2) complex numbers; and (3) efficient use of floating-point hardware.

Java does not support true multidimensional arrays. A true multidimensional array is characterized by an elemental data type, a rank (number of axes), and a rectangular shape defined by the extent of each axis. Java does support arrays of arrays, which allow it, to a certain degree, to simulate true multidimensional arrays. For example, a `double[ ] [ ]` object is an array of pointers to vectors of `doubles`. Note, however, that rectangularity of the structure is not guaranteed. Furthermore, Java arrays of arrays can display both intra- and inter-array aliasing. That is, two distinct indices for the same array can actually refer to the same data, and two (distinct or not) indices of different arrays can refer to the same data.

The solution we adopt, and also proposed by the Java Grande Forum, is to introduce multidimensional arrays in Java through a class library. To that purpose, we have developed the Array package for Java [7,30,31]. This package, written entirely in Java, provides a collection of classes for doing array-based numerical computing in Java. There is a specific class for each combination of rank and elemental data type. For example, `doubleArray2D` implements a two-dimensional rectangular array of `doubles`. This approach statically binds semantics to syntax, allowing a compiler to recognize the exact operation being performed. The shape of a multidimensional array from the Array package is defined at the moment the object is instantiated and it is immutable. The semantics of operations on Array package arrays is carefully defined so that compilers can aggressively optimize user code in a completely thread-safe manner.

A key compiler optimization that we have developed is versioning for bounds checking elimination [27,29]. This optimization creates regions of code that are guaranteed to be free of run-time exceptions. We can then apply a whole spectrum of traditional optimization and parallelization techniques that have been developed for more traditional languages (Fortran and C) during the past decades. A run-time test is performed during program execution to determine if it is safe, from a program semantics perspective, to execute this highly optimized version, or if a more conservative and slower version must be executed instead. We have found that, for numerical applications, the bulk of the computation is performed in the optimized versions. The combined approaches with the Array package and aggressive compiler optimizations have led to Java code executing at 80–100% of the speed of corresponding Fortran code.

Java does not support complex numbers as primitive types. The obvious solution is to implement a `Complex` class, and such a standard class is proposed by the Java Grande Forum. The problem with this approach is that a `Complex` object needs to be used to represent every complex number value created during a program execution, including intermediate results of expressions. This leads to a voracious rate of object creation and destruction, which in turn kills the performance of any complex number Java application. We have observed Java performance that is typically 100 times slower than a corresponding Fortran code.

We address the problem with complex numbers by enhancing our compiler with the ability to recognize operations on `Complex` objects. Using a technique called semantic expansion, the compiler replaces as many operations on `Complex` objects as possible with operations on complex values [40,41]. Complex values can be stored in machine registers and require no creation/destruction overhead. When combined with multidimensional arrays of complex numbers from the Array package, semantic expansion produces Java performance that is between 60 and 90% of the performance of corresponding Fortran codes.

With respect to efficient use of floating-point hardware, we focus on the fused multiply–add (fma) instruction. A fused multiply–add computes $a \times b + c$ with a single rounding operation at the end. That is, the result is the correctly rounded value for the infinitely precise number. For machines that support the fma, in particular the POWER/PowerPC, it can double the throughput of the floating-point units; however, the result from a fma instruction can be slightly different from that of a multiply-and-round followed by an add-and-round sequence. Therefore, the use of fmas is disallowed in Java. We have shown that the use of

fma can be very beneficial to the performance of Java codes. In the context of linear algebra codes, we have seen up to a factor of two improvements. In fact, only with the use of the fma we can approach Fortran performance for that class of computations.

We conclude by emphasizing that the technology for delivering high performance in Java numerical applications already exists. Standardization efforts are necessary to define "official" versions of the Array package and `Complex` class, so that Java environments can start to optimize for those components. We also need to promote the necessary changes in the language so that efficient use of floating-point hardware, in particular the fma instruction, can be made.

4.4. Bob Morgan viewpoint

High performance computing requires high performance everything—libraries, run-time systems, and compilers. Compiling for Java is a challenge. Just using libraries will not work—at some point a researcher will write different code that must be compiled; thus the compiler must generate code as good as Fortran—this is not easy in Java. In fact, it may not be possible. We are expending intense energy to build the best compiler [27].

4.5. John Parsons viewpoint

At ESI [5] we do not currently use Java in the development of our PREDICTIVE VIRTUAL PROTOTYPE TESTING products. Why, well, today 90% of the solver code is written in Fortran, which is still a very robust and powerful language. Currently all our developers are very proficient in Fortran and C. We draw many top PhD candidates to our development team but we encounter very few resumes from top candidates that have Java expertise. In the Graphics and User interface applications area we primarily use C and C++ in our legacy application but are quickly moving towards a new applications base that is primarily C++ combined with many high quality third party class libraries.

The solver development has been progressing for over 20 years in this way and we have hundreds of man-years of development that cannot be easily replaced. Recently many additional man-years have been spent moving the code towards supporting the multi-processor and massively parallel customer environments of today.

In fact, software development is only one part of this huge effort. The years of validation and testing with many very important companies around the world are an important part of our company's success. If we attempt to replace the software with JAVA, it would be very expensive. Any failure could be disastrous to both our company and that of our customers.

I do see that JAVA could be useful in some areas but improvements are required if it is to succeed as a replacement in Numeric Intensive Industrial Applications. Those areas in need of improvement include: Education and Training; Improved performance; Monetary support from UNIX vendors to port existing products; Support from the user community to port and validate our existing products re-engineered in JAVA; Re-use improvement to help in using existing Fortran and C code.

4.6. Thomas Wielenga viewpoint

Historically, numerical algorithms have been implemented in the language most often used by the practicing engineers. The first practical language to implement numerical algorithms was Fortran. It has a great deal of inertia, and the majority of public algorithms are still found in Fortran. However, there has been gradual progress to languages that are less immediately suited to numerical analysis. Gradually, algorithms became available in C (see Numerical Recipes in C), C++, and now Java. The penetration into the later languages has come at a faster pace than with the earlier languages. In addition, changes to these languages to facilitate efficient numerical calculations have come with time (Fortran), and are being proposed for Java (Java Grande Forum activity).

Object oriented languages are being supplanted by component architectures (COM, Java Beans, CORBA) as the building blocks of commercial programs. In addition to the component technologies, high level scripting languages (tcl, perl, python) are becoming popular as a means to "glue" various components together, to provide prototypes and even end-user solutions.

Although these new languages, component technologies, and scripting technologies are not especially suited to the fastest processing of numerical problems, the software industry is moving toward them as a solution to the even more difficult problem of providing overall software solutions on a variety of platforms. The provider of numerical solution methods should take the new component technologies into account when designing the interface to be used by its customers. Some of these technologies allow combining the strengths of these new software technologies with the speed of numerical algorithms implemented in traditional languages. Pros and cons of several approaches will be discussed.

5. Summary

During the presentations and the question and answer segment for this session, a variety of issues are examined. From the viewpoints summarized above in Section 4, we observe that although most panelists can see advantages of Java implementations of numeric-intensive, large-scale industrial simulations, they disagree as to the time frame when the limitations of Java for this category of application will be overcome. Some see the web-centric, object-oriented reusability attributes of Java as powerful motivation for immediate use. Others with lots of legacy Fortran,

C, and/or C++ code are more pessimistic with respect to their immediate needs for the fastest possible code execution and the lack of sufficient time and manpower to do significant code re-writes. As the availability of high quality and fast Java numerical libraries and tools increase, no doubt the pessimism of the latter group will subside. The question is when will this likely happen. Industrial people need to be vocal NOW if they want Java in the near future to meet there needs and replace their existing implementations. At present the vocal proponents seem to come primarily from academia, a few government labs, and some hardware/software vendors. Mainstream industrial people who could benefit from excellent Java implementations for large-scale computing seem to be mostly sitting on the sidelines waiting to see what happens.

Persons interested in continuing the dialogue about aspects of the panel theme should examine one or more of the internet references listed in the Reference section and/or contact any or all of the panel participants; participant addresses are given in Section 2 and their company web sites are listed in the bibliography.

References

[2] Center of Logistics and Expert Systems Publications, http://www.cle.de/cfd/publications/index.html/.

[3] Ninja: numerically intensive Java, http://www.research.ibm.com/ninja/.

[5] The ESI Home Page, http://www.esi.fr/.

[7] Artigas PV, Gupta M, Midkiff SP, Moreira JE. High performance computing in Java: language and compiler issues. IBM Research Report RC 21482 (96940), IBM T. J. Watson Research Center, Yorktown Heights, NY, 20 May 1999; published in Proceedings, 12th Workshop on Language and Compilers for Parallel Computers, August 1999 available at http://domino.watson.ibm.com/library/CYBERDIG.NSF/Home and then search under author; PostScript file available at http://www.research.ibm.com/ninja under author names.

[8] Bik A, Gannon DB. A note on Level 1 BLAS in Java. Proceedings, Workshop on Java for Computational Science and Engineering—Simulation and Modeling II, June 1997, available at http://www.npac.syr.edu/users/gcf/03/javaforcse/acmspecissue/latestpapers.html.

[10] Boisvert RF, Dongarra JJ, Pozo R, Remington KA, Stewart GW. Developing numerical libraries in Java. ACM 1998 Workshop on Java for High-Performance Network Computing, ACM SIGPLAN 1998; available at http://www.cs.ucsb.edu/conferences/java98.

[12] Casanova H, Dongarra JJ, Doolin DM. Java access to numerical libraries. Presented at the ACM 1997 Workshop on Java for Science and Engineering Computation, available at http://www.npac.syr.edu/projects/javaforcse/acmprog/prog.html.

[15] Getov V, Flynn-Hummel S, Mintchev S. High-Performance parallel programming in Java: exploiting native libraries. ACM 1998 Workshop on Java for High-Performance Network Computing; available at http://www.cs.ucsb.edu/conferences/java98/program.html.

[16] Ginsberg M. Current and future status of HPC in the world automotive industry. In: Henderson ME, Anderson CR, Lyons SL, editors. Object oriented methods for inter-operable scientific and engineering computing, Philadelphia: SIAM, 1999. p. 1–10.

[17] Ginsberg M. Influences, challenges, and strategies for automotive HPC benchmarking and performance improvement. Parallel Computing Journal 1999;25(12):1459–76.

[18] Ginsberg, M. Future directions of Java in support of automotive large-scale simulations. Proceedings, Computer Technology Solutions for the Manufacturing Enterprise, Society of Manufacturing Engineers, Dearborn MI, September 1999.

[19] Gosling, J. The evolution of numerical computing in Java. http://java.sun.com/people/jag/FP.html.

[20] Gray MG, Roberts RM. Object-based programming in Fortran 90. Computers in Physics 1997;11(4):355–61.

[23] Recent activities of the Java Grande Forum numerics working group, http://math.nist.gov/javanumerics/reports/jgfnwg-02.html.

[24] Java Grande Forum Reports. Making Java work for high-end computing. JGF-TR-1; Desktop access to remote resources. JGF-TR-2; MPI for Java position document and draft specification. JGF-TR-3 available at http://www.javagrande.org/reports.htm.

[25] Kahan W, Darcy JD. How Java's floating-point hurts everyone everywhere. ACM 1998 Workshop on Java for High-Performance Network Computing, Stanford University, Palo Alto, CA, 1 March 1998; http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf.

[26] Midkiff SP, Moreira JE, Snir M. Java for numerically intensive computing: from flops to gigaflops, Proceedings of Frontiers '99. Los Almitos, CA: IEEE, Computer Society Press, 1999. p. 251–9.

[27] Midkiff SP, Moreira JE, Snir M. Optimizing array reference checking in Java programs. IBM Systems Journal 1998;37(3):409–53.

[28] Moreira JE, Midkiff SP, Gupta M. A comparison of Java, C/C++, and Fortran for numerical computing. IEEE Antennas and Propagation Magazine 1998;40(5):102–5.

[29] Moreira JE, Midkiff SP, Gupta M. From flop to megaflops: Java for technical computing. Proceedings of the 11th International Workshop on Languages and Compilers for Parallel Computing, LCPC'98, 1998; also IBM Research Report RC 21166 (revised) (94594), 31 August 1998; available at http://domino.watson.ibm.com/library/CYBERDIG.NSF/Home and then lookup author(s) or report name.

[30] Moreira JE, Midkiff SP, Gupta M. A standard Java array package for technical computing. IBM Research Report RC 21369 (96233), IBM T.J. Watson Research Center, Yorktown Heights, NY, 21 December 1998. Proceedings of the 1999 SIAM Conference on Parallel Processing for Scientific Computing. http://domino.watson.ibm.com/library/CYBERDIG.NSF/Home and then search under author.

[31] Moreira JE, Midkiff SP, Gupta M, Artigas PV, Snir M, Lawrence RD. Java programming for high performance numerical computing, IBM Research Report RC 21481 (96939), IBM T.J. Watson Research Center, Yorktown Heights, NY, 20 May 1999; to appear in IBM Systems Journal, 2000; http://domino.watson.ibm.com/library/CYBERDIG.NSF/Home and then search under author.

[32] Moreira JE, Midkiff SP, Gupta M, Lawrence RD. High performance computing with the Array Package for Java: a case study using data mining. Proceedings of SC99. Portland, Oregon, November 1999.

[33] Moreira JE, Midkiff SP, Snir M. A Java array package. See http://math.nist.gov/javanumerics/array/. Available for download at www.alphaWorks.ibm.com/tech/ninja.

[34] Morgan R. Building an optimizing compiler. Digital Press, 1998.

[38] Sun Microsystems. Sun proposes modification to Java programming languages' floating point specification. 1998; see http://www.sun.com/smi/Press/sunflash9803/sunflash.980324.17.html.

[40] Wu P, Midkiff SP, Moreira JE, Gupta M. Efficient support for complex numbers in Java. Research Report RC 21393, IBM T.J. Watson Research Center, Yorktown Heights, NY, 27 January 1999. Proceedings of the 1999 ACM Java Grande Conference.

[41] Wu P, Midkiff SP, Moreira JE, Gupta M. Improving Java performance through semantic inlining. Technical Report RC 21313 (96030), IBM Research Division, IBM T.J. Watson Research Center, Yorktown Heights, New York, 14 October 1998; available at http://domino.watson.ibm.com/library/CYBERDIG.NSF/Home and then search under author(s) or title.

Further reading

[1] HPC Research and Education Home Page, http://www.rust.net/~ginsberg.

[4] Compaq's Extreme Java TechnologyHome Page, http://www.digital.-com/java/.

[6] Engineering Insight, LLC Home Page, http://www.EngInsight.com.

[9] Blount B, Chatterjee S. An evaluation of Java for numerical computing. Proceedings of ISCOPE'98, Lecture notes in computer science, vol. 1505. 1998. p. 35–46.

[11] Budimlic Z, Kennedy K. *Optimizing Java: theory and practice*. Concurrency: Practice and Experience 1997;9(6):445–63.

[13] Cierniak M, Li W. Just-in-time optimization for high-performance Java programs. Concurrency, Practice and Experience 1997;9(11):1063–73.

[14] Compaq's fast Java virtual machine, June 1999, http://www.digital.-com/java/FastJVM.html.

[21] Hauser J. JavaPar: Internet based parallel computational fluid dynamics solver using the Java thread concept. See http://www.cle.de/cfd/personnel/haeuser/index.html.

[22] Hauser J, et al. A pure Java parallel flow solver. Proceedings, 37th AIAA Aerospace Science Meeting and Exhibit, AIAA 99-0549, Reno, NV, 11–14 January 1999; also available at http://www.cle.de/cfd/publications/HaveJava1.pdf.

[35] Philippsen M. Is Java ready for computational science? In Euro-PDS'98, Second European Parallel and Distributed Conference, Vienna, Austria, 1–3 July 1998. p. 299–304; also at http://www.ipd.ira.uka.de/~phlipp/JavaCS.ps.gz.

[36] Schwab M, Schroeder J. Algebraic Java classes for numerical optimization. ACM Workshop on Java for high-performance network computing. ACM SIGPLAN, 1998. See http: //www.cs.ucsb.edu/conferences/java98.

[37] Seshadri V. IBM high performance compiler for Java. AIXpert Magazine September 1997; see http://www.developer.ibm.com/library/aixpert.

[39] Winkelmann R, Häuser J, Williams RD. Strategies for parallel and numerical scalability of large CFD codes. In: Tezduyar TE, editor. To be published in: special issue of Parallel computing. Amsterdam: North-Holland, 1998.