Java and Web Technologies for Simulation and Modelling in Computational Science and Engineering

Geoffrey C. Fox* Wojtek Furmanski†

Abstract

We discuss the role of Java and Web technologies for general simulation. We classify the classes of concurrency typical in problems and analyze separately the role of Java in user interfaces, coarse grain software integration, and detailed computational kernels. We conclude that Java could become a major language for computational science, as it potentially offers good performance, excellent user interfaces, and the advantages of object-oriented structure.

1 Introduction

The World Wide Web provides important infrastructure for scientific and engineering computation. The distributed computing hardware of the Web has remarkable potential compute performance—1,000 times that of the largest supercomputer. This ratio largely reflects the ratio of monetary investment in the two fields. Of course, the Web does not support the low latency and high bandwidth required by most parallel simulations. However, we believe that an attractive scientific computing environment can be built on top of Web software by adding to the basic Web loosely coupled distributed computing model, the necessary added functionality for computational science. We analyze, in Section 2, the various forms of concurrency seen in applications, and then in the last three sections, discuss three major areas where Java can be effectively used. We conclude that Java could well become a dominant language in science and engineering.

2 Concurrency in Applications

In understanding the role of the web in large-scale simulations, it is useful to classify the various forms of concurrency in problems into four types [5].

1. **Data Parallelism**

   This is illustrated by natural parallelism over the particles in a molecular dynamics computation; over the grid points in a partial differential equation; over the random points in a Monte Carlo algorithm. In the Web computation of the factors of RSA130 [7], we can consider the parallelism over possible trials in the Sieve algorithm as the "data" for data parallelism in this application. Data parallelism tends to be "massive" because computations are typically time consuming due to a large amount of data.

*Syracuse University, Northeast Parallel Architectures Center, 111 College Place, Syracuse, New York 13244, gcf@npac.syr.edu

†Syracuse University, Northeast Parallel Architectures Center, 111 College Place, Syracuse, New York 13244, furm@npac.syr.edu

Thus, data parallelism is parallelism over what is "large" in the problem. It is not difficult to find data parallel problems today with parallelism measured in the millions (e.g., a $100 \times 100 \times 100$ grid) and by the year 2007, billion-way data parallelism can be expected.

2. **Functional Parallelism**

   Here we are thinking of typical thread parallelism, such as the overlap of computation (say, decompressing an image) and communication (fetching HTML from a server). More generally, problems typically support overlap of I/O (disk, visualization) with computation. We also, of course, can have multiple compute tasks executing concurrently. This form of parallelism is present in most problems; the units are modest grain size (larger than a few instructions scheduled by a compiler, smaller than an application), and typically not massively parallel. Further, such functional parallelism is typically implemented using a shared memory and, indeed, its existence in most problems makes few way parallel shared memory multiprocessors very attractive.

3. **Object Parallelism**

   We could mean many things by this, but we have in mind the type of problems solved by discrete event simulators. These are illustrated by military simulations where the objects are "vehicles," "weapons," or "humans in the loop." The well-known SIMNET or DSI (Distributed simulation Internet) have already illustrated the relevance of distributed (Internet) technology for this problem class [8]. Object descriptions are similar to data parallelism except that the fundamental units of parallelism, namely objects are quite large, corresponding to a macroscopic description of an application. Thus, a military battle is described in terms of the units of force (tanks, soldiers) with phenomenological interactions rather than in (unrealistic in this case) fundamental description in terms of atomic particles or finite element nodes. For a typical "data parallel" problem, the fundamental units of parallelism (grid points) are typically smaller.

4. **Metaproblems**

   This is another functional concurrency, but now with large-grain size components. In image processing, one often sets up an analysis system where the pixels are processed by a set of separate filters—each with a different convolution or image understanding algorithms. Software systems, such as AVS and Khoros are well-known tools to support such linked modules. So a metaproblem is a set of linked problems (databases, computer programs) where each unit is essentially a complete problem itself. Dataflow (a graph specifying how problems accept data from previous steps and produce data for further processing) is a successful paradigm for metaproblems. In manufacturing, one often sees metaproblems as building a complex system, such as an aircraft, requiring linking airflow, controls, manufacturing process, acoustic, pricing and structural analysis simulations. It has been estimated that designing a complete aircraft could require some 10,000 separate programs—some complicated ones such as airlow simulation were mentioned above, but as well there are simpler but critical expert systems to locate inspection ports, and other life-cycle optimization issues [3], [4]. Metaproblems have concurrency that it typically quite modest. They differ from the examples, in category 2 above, in that the units have larger grain size and are more self contained. This translates into different appropriate computer architectures. Modest grain size functional parallelism (2) needs low latency and high bandwidth communication—typically implemented with a shared memory. Metaproblems are naturally implemented in a distributed (Web) environment—latency is often unimportant while needed network bandwidths are more variable.

3 Overview of Web and Parallel Computing Software Issues

We can view computing (as many other enterprises) in terms of a pyramid with widely deployed cheap systems at the bottom of the pyramid, and the few high-performance systems as the top. There is much more computing power in the distributed collection of consumer-oriented products—PCs, videogames, Personal Digital Assistants, Digital Set Top boxes, etc. This dominant dollar investment in the consumer products implies that one can expect the bottom of the pyramid to have much better software than the top. Software investment must be roughly proportional to market size, and so we see PCs, workstations, and MPPs (Massively Parallel Processors) offering increasing unit software price and decreasing software quality and functionality. The Web, perhaps, offers now the best available software (as it is potentially the largest market). When the PC market dominated quality consumer software, it was hard for the parallel processing community to take advantage of it. PCs offer, of course, a sequential computer model, but now the Web software targets a very rich distributed computing model. It seems to us clear that we can, and indeed must, build MPP software with a backbone architecture of Web software. As mentioned in the Introduction, we can then view parallel processing as a special case of a distributed model with stringent synchronization constraints. We view this as leading to a set of Compute Webs, which we describe in the following sections.

This approach has the added advantage that we can build Compute Webs by either running Web clients or servers with synchronization/compute enhanced Web software, or use the latter software to provide a very attractive user environment on specialized MPPs whose low latency and high-bandwidth communication enable critical parallel computations.

In the following, we discuss the role of Web hardware and, especially, software for three distinct parts of computation.

1. User (client) view—problem specification, visualization, computational steering, data analysis

2. Metaproblem implemented on a distributed computer

3. Individual computationally complex components of the metaproblem implemented on high-performance computers, which could in fact be a distributed system itself.

We cover these three parts—graphical user interface, dataflow for metaproblems/software integration, and hardcore computation, in the next three sections.

4 WebWindows and the User View

We abstract future high-performance computing environments into four layers detailed below [9].

a) Fully visual or scripted (interpreted) environment exhibiting domain specific functionality

This is the typical graphical interface allowing manipulation at either metacomputer, or individual component level.

b) Partially scripted level offering

Portable flexible programming at some performance cost - illustrated by Java in applet mode

c) Traditional compiled level

Offering a high-level language with few machine dependent features, and getting high performance—traditionally within about a factor of two of the peak performance possible on the particular algorithm—illustrated by coupled Fortran, C, C++, and Java.

d) Traditional machine specific level

Rarely used by application programmers or even those building (high level) tools. Clearly, allows user to obtain peak performance at the cost of a very inconvenient programming environment.

Levels c) and d) include the computationally intense parts of the problem, which can be implemented on appropriate servers. However levels a) and b) which we discuss in this section, are likely to be executed in the client machine/environment. We describe the current trends in software strategy [2], [6] as a shift from software built in terms of PC Windows, Macintosh, UNIX environments to a WebWindows basis, i.e., software built on the interfaces defined by Web servers and Web clients. This is, of course, a valid approach whether one is writing for a single stand-alone machine (running a Web server and client) or the entire worldwide network. In this sense, the use of Web technology for user interfaces is trivial—the user interface is not constrained greatly by the difficulties of high-performance computation, as it runs on the "conventional" client side and so can naturally use best client side technologies. Some examples of Web based user interfaces are:

- NCSA's biology workbench [10], which is a CGI interface to a collection of useful computational biology resources.
- An environment [11] built by Gregor von Laszewski to support a metaproblem—the linked components of a large scale NASA weather simulation. This uses a Java graphical editor to allow the user to choose which program component to run on which of a distributed set of computers.
- NIST's user interface [12] for their IBM SP2 parallel computer.
- The Virtual Programming Laboratory (VPL) [13] built by Kivanc Dincer and used in the Syracuse course, CPS615, this semester to support parallel program development.
- A typical Java visualization applet [14] to support VPL.

We expect this type of interface development to continue and become the norm. However, we see a particularly important role for Java (and VRML) in terms of level b). Namely, Java seems an attractive language for building client side data analysis systems. These typically involve both computation and visualization—in which linkage, Java has unique capabilities. Thus, we expect a set of high quality Java applets (or compiled plug-ins) to be developed which support this analysis. Those applets will be used at level a) by the general user with the expert modifying the code of the applets (level b)) for customized capability. A good example of Java for scientific visualization is the work of Cornell [15] on an applet for teaching fracture mechanics. We depict the resultant, environment which essentially becomes a Java wrapper for code written in traditional languages and running on sequential, parallel or distributed computers. This use of Java is likely to grow rapidly as it requires modest changes to existing software and adds great value without changing the familiar programming paradigm. However, we see it as a natural Web "seed" that can grow into the more pervasive use of Java.

5 WebFlow and Coarse Grain Software Integration

As we have discussed, it is very natural to use web hardware and software to implement control of metaproblems [16]. Although we only described earlier the dataflow model for this, one can, of course, use these ideas for any application with linked components that have relatively large chunks of computation that dwarf the latency and bandwidth implied by using the Web as a compute engine. In fact, we can include our recently completed RSA130 factoring project [17] in this class. This distributed the sieving operations over a diverse range of clients (from an IBM SP2 at NPAC to a 386 laptop in England) under the control of set of servers. This was implemented as a set of Web server CGI Perl scripts FAFNER [7]. These created daemons to control the computation on each client which returned results to the server that accumulated results for final processing to locate factors.

We can extract two types of computing tasks from our factorization experience [1]. The first is the resource management problem—identifying computer resources on the Web; assigning them suitable work; releasing them to users when needed, etc. A sophisticated Web system ARMS [18] for this is being developed by Lifka at the Cornell Theory Center. Well-known distributed computing systems in this area include LSF, DQS, Codine and Condor (see review in [19]), and this seems a very natural areas for the use of Web systems including linked databases to store job and machine parameters.

The second task is the actual synchronization of computation within a given problem—resource management, on the other hand assigns problems to groups of machines and does not get involved with detailed parallel computing algorithm and synchronization issues. Here, we see two general concepts. One is support of the messaging between individual nodes that creates a virtual (parallel) machine out of the World Wide Web.

This low level support is called WebVM and should implement the functionality of parallel systems, such as MPI in terms of Web technology message systems—either HTTP or direct Java server—server (client) connections. Here, the most elegant model is perhaps based on a mesh of Web Servers [20] although today's most powerful implementations would use like FAFNER, a mesh of Web clients controlled by a few servers [21]. In the spirit of WebWindows, we can expect servers or server equivalent capability to become available on all Web connected machines. Note that the natural Web model is server-server, and not server-client and indeed this supports the traditional NII dream of democracy with everybody capable of either publishing or consuming information.

On top of WebVM, one can build higher level systems, such as the distributed shared memory model (called WebHPL) or more easily an explicit message passing system, such as the dataflow model. WebFlow supports a graphical user interface ([1], [2], [16]) specifying metaproblem component linkage and one can naturally design domain specific problem solving environments in this framework. One would support scripted "little languages" (designed for each application) at the top level a) (in classification of Section 4, which would allow for more flexible and dynamic metaproblem component linkage.

Now is, of course, a confusing time for there are as many compute-web implementation strategies as there are major players in emerging Web technology—especially as we evolve from powerful, but rather ad hoc server side CGI scripts to integrated dynamic Java client and server systems. Thus, now is not the time for "final solutions" but rather for experimentation and flexibility to examine and influence the key building blocks of future Web computers.

Finally, note that the Web encourages new models for computation with problems publishing their needs and Web compute engines advertising their capabilities and dynamic matching of problems with compute resources.

6 Java as the Language for Computational Science and Engineering

We recently held a workshop [22] on this theme at Syracuse. This covered generally the topics of the last two sections where we saw Java as clearly attractive for both user interfaces, wrappers, and the metaproblem control. Here, we consider its possible role as the basic programming language for science and engineering - taking the role now played by Fortran 77, Fortran 90, and C++.

Java's most important advantage over other languages is that it will be learnt and used by a broad group of users. Java is already being adopted in many entry level college programming courses and will surely be attractive for teaching in middle or high schools. Java is a very social language as one naturally gets Web pages from one's introductory Java exercises that can be shared with one's peers. We have found this as a helpful feature for introductory courses. Of course, the Web is the only real exposure to computers for many children, and the only languages they are typically exposed to are Java, JavaScript, and Perl. We find it difficult to believe that entering college students, fresh from their Java classes, will find it easy to accept Fortran, which will appear quite primitive in contrast. C++ as a more complicated systems building language may well be a natural progression, but although quite heavily used, C++ has limitations as a language for simulation. In particular, it is hard for C++ to achieve good performance on even sequential and parallel code, and we expect Java not to have these problems.

In fact, let us now discuss performance, which is a key issue for Java. We have already suggested a multilevel scientific programming environment that would use purely scripted, applet mode and purely compiled environments with different tradeoffs in usability and performance. As discussed at our workshop, there seems little reason why native Java compilers, as opposed to current portable JavaVM interpreters or Just in Time compilers (JIT), cannot obtain comparable performance to C or Fortran compilers. A major difficulty is the rich exception framework allowed by Java that could restrict compiler optimizations. Users would need to avoid complex exception handlers in performance critical portions of a code.

An important feature of Java is the lack of pointers and their absence, of course, allows much more optimization for both sequential and parallel codes. Optimistically, we can say that Java shares the object oriented features of C++ and the performance features of Fortran.

One interesting area is the expected performance of Java interpreters (using just in time techniques) and compilers on the Java bytecodes (Virtual Machine). Here, we find today perhaps a factor of 4–10 lower performance from a PC JIT compiler compared to C compiled code. Consensus at the workshop expected this performance degradation to be no worse than a factor of two for the portable applet mode. As described above, with some restrictions on programming style, we expect Java language or VM compilers to be competitive with the best Fortran and C compilers. Note that we can also expect a set of high performance "native class" libraries to be produced that can be down loaded any accessed by applets to improve performance in the usual areas one builds scientific libraries.

One interesting omission is a purely interpreted version of Java—level a). This would also be very helpful for teaching. JavaScript is interpreted, but we would view it as a "little language" for document handling - and not a general Java-like interpreted environment.

Finally, we will discuss parallelism in Java. Here, we return to the four categories of concurrency.

1. **Data Parallelism**

   This is supported in Fortran by either high level data parallel HPF or at a lower level Fortran plus message passing (MPI). Java does not have any built in parallelism of this type, but at least the lack of pointers means that natural parallelism is less likely to get obscured. There seems no reason why Java cannot be extended to high level data parallel form (HPJava) in a similar way to Fortran (HPF) or C++ (HPC++) [23]. At the lower message passing level, the situation is clearly satisfactory for Java as the language naturally supports inter-program communication, and the standard capabilities of high-performance message passing are being implemented for Java [24].

2. **Modest Grain Size Functional Parallelism**

   This is built into the language with threads for Java and has to be added explicitly with libraries for Fortran and C++.

3. **Object Parallelism**

   This is quite natural for C++ or Java where the latter can use the applet mechanism to portably represent objects. We have built a collaboration system TANGOsim where a Java server controls a set of Java applets and other applications spawned from them [25]. We generalized the session manager present in collaborative systems to be a full event driven simulator. This illustrates the power of Java for this problem class and shows that it can unify traditional time stepped simulations (typical for data parallelism) with event driven forces modeling, and other such simulations.

4. **Metaproblems**

   We have already discussed in Section 5, the power of Java in this case for overall coarse grain software integration.

In summary, we see that Java has no obvious major disadvantages and some clear advantages compared to C++ and especially Fortran as a basic language for large scale simulation and modeling. Obviously, we should not and cannot port all our codes to Java. Rather, we can start using Java for wrappers and user interfaces. As compilers get better, we expect users will find it more and more attractive to use Java for new applications. Thus, we can expect to see a growing adoption by computational scientists of Web technology in all aspects of their work.

References

[1] Fox, G. C., Furmanski, W., Chen, M., Rebbi, C., and Cowie, J. H. "WebWork: integrated programming environment tools for national and grand challenges." Technical Report SCCS-715, Syracuse University, NPAC, Syracuse, NY, June 1995. Joint Boston-CSC-NPAC Project Plan to Develop WebWork.

[2] Fox, G. C., and Furmanski, W. "The use of the national information infrastructure and high performance computers in industry," in *Proceedings of the Second International Conference on Massively Parallel Processing using Optical Interconnections*, pages 298–312, Los Alamitos, CA, October 1995. IEEE Computer Society Press. Syracuse University Technical Report SCCS-732.

[3] Fox, G. C. "High performance distributed computing." Technical Report SCCS-750, Syracuse University, NPAC, Syracuse, NY, December 1995. To appear in Encyclopedia of Computer Science and Technology.

[4] Fox, G. C. "A tale of two applications on the NII." Technical Report SCCS-756, Syracuse University, NPAC, Syracuse, NY, March 1996. Submitted to the 1996 Sixth Annual IEEE Dual-Use Technologies and Applications Conference.

[5] Fox, G. C. "An application perspective on high-performance computing and communications." Technical Report SCCS-757, Syracuse University, NPAC, Syracuse, NY, April 1996.

[6] Fox, G. C., and Furmanski, W. "SNAP, Crackle, WebWindows!." Technical Report SCCS-758, Syracuse University, NPAC, Syracuse, NY, April 1996.

[7] "Factoring RSA 130 on the Web," http://www.npac.syr.edu/factoring

[8] "Defense Modeling and Simulation Office," http://www.dmso.mil/

[9] "Suggested Software Environments in the Year 2007," http://www.npac.syr.edu/users/gcf/petastuff/petasoftwp

[10] "NCSA Computational Biology Workbench," http://bioweb.ncsa.uiuc.edu

[11] "Web based Distributed Computing Environment for Data Assimilation," http://www.npac.syr.edu/projects/nasa/home.html

[12] "NIST's SP2 Interface," http://www.itl.nist.gov/div895/sasg/websubmit/

[13] "HPF and MPI Virtual Programming Language," http://www.npac.syr.edu/projects/cps615fall96/

[14] "Web based Visualization," http://www.npac.syr.edu/users/dincer/pablo/

[15] "Java Applet for Crack Propagation," http://www.msc.cornell.edu/ houle/cracks

[16] "NPAC Projects in Web based HPCC," http://www.npac.syr.edu/projects/webbasedhpcc

[17] "Tutorial on RSA 130 Factoring," http://www.npac.syr.edu/users/gcf/crpcrsamay96

[18] "Discussion of ARMS Resource Management," http://www.tc.cornell.edu/er96/ff04fall96/ffo1arms.html

[19] "Review of Cluster Management Software," http://nhse.cs.rice.edu/NHSEreview/96-1.html

[20] "Talk on WebFlow and WebVM," http://www.npac.syr.edu/projects/webspace/doc/hpdc5/hpdc5.html

[21] "Super-Web: Towards a Global Web based Parallel Computing Infrastructure," http://www.cs.ucsb.edu/ schauser/papers/96-superweb.ps
"Create Your Own Supercomputer with Java," http:/www.javaworld.com/javaworld/jw-01-1997/jw-01-dampp.html

[22] "Workshop on Java for Computational Science," http://www.npac.syr.edu/projects/javaforcse

[23] "Data Parallel Java Prototype," http://www.npac.syr.edu/users/dbc/HPJava

[24] "GLOBUS Metacomputing Infrastructure," http://www.mcs.anl.gov/globus

[25] "TANGO Java Collaboratory," http://www.npac.syr.edu/projects/tango