Java Grande: Software Infrastructure for HPCC

Geoffrey Fox **gcf@npac.syr.edu**
NPAC Syracuse University
111 College Place
Syracuse NY 13244-4100

Abstract

We describe the definition, motivation and current status of Java Grande activities. We introduce 3 roles of Java in Grande programming at client, middleware or backend tiers of a computing system. We start with Java as a language and describe where it is clearly good and where it could be good! The Java Grande Forum has numerical and distributed computing working groups and projects include the study of changes to Java and its runtime to enhance Grande applications and their programming environment community. There is an important activity to define seamless interfaces allowing universal access to general hosts. Benchmarks for all sorts of Grande applications are critical. We discuss Java for Parallel Computing including message passing (MPI) and data parallelism.

1 Use of Java in Technical Computing

For some years, my research has been influenced by the difficulties of turning good research ideas in HPCC into quality deployable software. We believe that it is perhaps impossible for our field to develop sustainable high quality programming environments without taking more advantage of commodity software systems. HPCC as a field is perhaps 1% of the total computing activity but has probably the hardest problems to solve with large applications exhibiting critical performance requirements. We note that the commodity Web and distributed object communities have produced a rich software infrastructure. This is of direct relevance as it supports with open interfaces a complete distributed computing model – something not contained in previous mainstream computing systems such as those of standalone PC's or those still used to run IBM Enterprise Information servers. Our plan is to focus HPCC software on only those areas where there are special high performance needs. For instance we view parallel computing as just a "special case" of a distributed system requiring low latency and high bandwidth. Whether or not you fully agree with this approach, many researchers consider it interesting to look at the opportunities of using Web, distributed object and/or Java technologies in HPCC. We consider the latter here and describe the Java Grande process and its initial results. Our understanding of these issues has been influenced by the work reported at four workshops held in Syracuse (Dec. 96), Las Vegas (June 97), Palo Alto (February 98) and Southampton (England, September 98). [1-4] We refer the reader to our papers on the "Pragmatic Object Web" for a more general discussion of the relevance of other emerging commodity software technologies.[5-8] Java is a little more general than one might think. For instance, the use of CORBA might in fact require our study of high performance Java as the CORBA broker is quite likely to be written in Java. In general we except Java to underlie much important distributed object and web software infrastructure.

In Fig. 1, we picture three possible roles for Java corresponding to its use respectively in the client, middle-tier or backend tiers of a computing environment. The use of Java at the client is well understood and in spite of the battle between Microsoft and Netscape, seems likely to flourish with some competition from increasing use of dynamic HTML. The use of Java at the middle (server) tier is perhaps the dominant use of Java in the Intranet software industry and Java is attractive here because it is a very productive programming environment. C++ currently gives higher performance but Java servers are already very robust and new generations of Java compilers are expected to give excellent performance. Finally there is the third role of Java – that as a basic programming environment. Here performance is also critical and involves special consideration because of the importance of floating point arithmetic, which is less relevant in the middle tier.

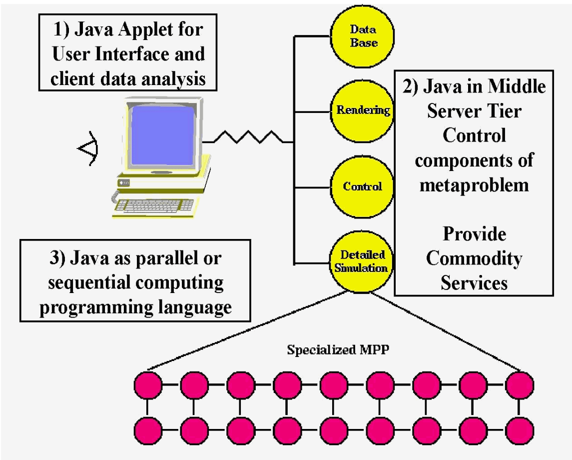

Figure 1: *Three Roles of Java in HPCC*

2 Why Explore Java as a Technical Computing Language?

Why would one use Java to code one's favorite scientific code? One can cite the usual features. Firstly, the Java Language has several good design features. It is in particular secure, safe (with respect to bugs), object-oriented, and familiar (to C, C++ and even Fortran programmers). Secondly, Java has a very good set of libraries covering everything from commerce, multimedia, images to math functions (see for instance NIST's library under development at [9-10]). These frameworks can embody both real software (i.e. fully coded methods) but also interfaces which define standards that lead to uniform easier to use uniform environment where different vendors can focus on the best implementation of a particular service without using their own irrelevant proprietary names. It is curious that the Fortran community never agreed on the names of basic numerical libraries – we should spend our creative energy on the best coding of the best FFT algorithm and not on the subroutine calling sequence.

Java is regarded as a very productive programming environment. Partly this reflects that industry is aggressively developing for Java the best integrated program development environments. However the productivity of Java also reflects its natural integration with network and universal machine which supports its powerful "write once-run anywhere" model

Again, Java has best available electronic and paper training and support resources. There are in particular over 1000 books in Java. There is a large and growing trained labor force. Java is already being adopted in many entry-level college programming courses. In fact, it seems likely that in the future, students entering college will increasingly have Java expertise. They will learn about it from their excursion on the Internet while it should grow in interest as the language used to teach programming at middle and high schools. (Note NPAC's Java Academy at [11]) which was very successfully taught each Saturday in the depth of a bleak Syracuse winter to a group of middle and high school students and teachers). Java's natural graphical (applet) view makes it a very social language whose value is more obvious to a beginning programmer than C++ or Pascal. Of course, the Web is the only real exposure to computers for many children, and the only languages, to which they are typically exposed today, are Java, JavaScript, and Perl. We find it difficult to believe that entering college students, fresh from their Java classes, will be willing 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 if the process described in sec. 3 is successful.

As well as these laudable positive features of Java, we can also compare Java with the "competition" which is Fortran or C++ for serious technical computing.

Fortran77 has excellent compilers, good user base but will not be taught broadly and clearly limited in capabilities; in particular it is not object oriented. It appears that although Fortran90 and HPF have useful features, they have not "taken off" and it seems likely that they will not "make it"

Five years ago, it looked as though C++ could become language of choice in complex scientific codes (perhaps with Fortran as inner core and especially when irregular data structures could not be easily expressed in Fortran). However this movement appears stalled – partly because this trend was halted by a growing interest in Java and users are awaiting events. C will remain widely used as a simple elegant language but object oriented techniques seem essential in large software systems and so the essential competition appears to lie between C++, Fortran90 and Java. Note that the C++ language is complex and splintered with no agreement on standards for libraries and parallelism. This is partly because its use in Grande applications is too small to motivate standards and partly due to the prevailing culture.

So we argue that although existing large-scale codes are written in Fortran C and C++, the associated unattractive and comparatively unproductive programming environment handicaps developers. Current languages and tools are sufficient but it does not seem likely that one can easily greatly improve on existing environments without a radically new approach. We suggest that it will easier to try to build an attractive technical computing environment around Java rather than the existing languages.

We can list some additional reasons why we might be more successful in Java than previous Fortran or C++ based programming environments. Firstly, Java has some natural advantages due its Internet base with threads and distributed computing built in. Secondly, Java is a young language and we can take steps now to avoid unproductive proliferation of libraries and parallel constructs. We could be third (Fortran, C++, and now Java) time lucky. More seriously, if our proposed changes, described later, are adopted, Java should exhibit the expressivity and object oriented advantages of C++ combined with performance levels of C and Fortran. Finally we note that one can use Java's clear advantages in building user interfaces as an entrée into its use in other aspects of large-scale programming.

However there are some serious problems to be solved if Java is to realize its potential in technical computing and now we turn to discuss these.

3 Java Grande

First we need to define a Grande application as any sort of large-scale or technical commercial or academic problem. Thus it subsumes areas such as High Performance Network Computing or HPDC (High Performance Distributed Computing); Scientific and Engineering Computation; (Distributed) Modeling and Simulation; Parallel and Distributed Computing; Data Intensive Computing; Communication and Computing Intensive Commercial and Academic Applications; High Performance Computing and Communication (HPCC); and finally Computational Grids.

We adopted this offbeat nomenclature, as it was hard to find a "conventional name" that doesn't get misunderstood by some community. Now Java Grande is the application of Java to Grande applications; Grandecomputers are of course compute engines used to execute Grande codes and the adjective can be used in other related ways.

The Java Grande forum was set up to enhance the possibility that one can build around Java a better Grande application environment than is available through Fortran or C++. We described in the previous section why this might be possible and the Forum's sole goal is sponsor community activities designed to realize the "best ever Grande programming environment". The Forum products include recommendations and community actions that could lead to necessary changes to Java or establishment of standards (frameworks) for "Grande" libraries and services. We have had internal meetings in March, May and August 1998 and a public discussion of our initial conclusions at SC98 in Orlando on November 13 98. The current status is given at our home page [12] while the NPAC resource[13] has more personal broader collection. The Forum is interacting in two rather different dimensions. In the near term, we need to work with the computing mainstream and Sun to discuss a few key changes in Java to allow it to be a complete efficient Grande Programming Language. This includes the floating-point processing, complex type and RMI performance issues described later.

Secondly The Forum needs to work within the Grande community to encourage and initiate those activities that will lead to standards in such areas as numeric libraries and the Seamless Computing Interface. We suggest that the Grande community has unnecessarily handicapped progress by having as much creativity in the interfacing of its artifacts as in the essential algorithms. As we illustrate in the next section for databases, sometimes all can benefit if one agrees to standard which initially handicap particular and perhaps even the best implementations.

The Forum is set up currently with two major working groups. The *Numerics* working group is led by Ron Boisvert and Roldan Pozo from NIST and is centered on Java as a language for mathematics. Issues studied include changes in Java controversial handling of floating point, which currently has goal of reproducible results, but this leads to non-optimal accuracy. They address the support for efficient Complex types or classes. This can be implemented with lightweight classes and operator overloading, which can also be applied in other circumstances. Support for "Fortran rectangular multidimensional arrays" is important as Java's current multi-subscript language construct gives "arrays of arrays", which often do not lead to efficient code. Another major activity is the design and implementation of high quality math libraries with agreed interfaces – examples are FFT, Matrix algebra, and Transcendental functions.

Performance and expressivity and their tradeoff underlie these proposed enhancements. As discussed in the four workshops on Java for Science and Engineering computation [1-4], the goal is Java compiler's that obtain comparable performance to those for C or Fortran. Marc Snir has given a very clear analysis [14] of the different issues that inhibit the performance of Java on classic array-based scientific codes. Industry efforts are mainly focussed on Just in Time compilers (JIT) which support the critical applet and servlet models of computation. However traditional native machine specific compilers are possible and will surely be useful. It will be interesting to compare their performance with the best JIT's and see if and for what application any performance degradation for servlets and applets outweighs the convenience of their mobile dynamic portable computing model. A related issue is if the Java language version of a code has any more information for a native or JIT compiler than the VM (or Java bytecode) representation.

Initial studies suggest that the VM and language versions of the code can be compiled with comparable performance. Difficulties in compiling Java [15,16] include quite surprising points such as Java's rich exception framework 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 significantly more optimization for both sequential and especially parallel codes. In general with some possible restrictions on programming style, we expect Java 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 downloaded and accessed by applets to improve performance in the usual areas one builds scientific libraries.

The charter of the second working group led by Dennis Gannon and Denis Caromel (INRIA, France), includes issues associated with coarse grain distributed scientific objects; distributed and parallel computing, metacomputing [17-20], *concurrency* support and *applications*. The detailed agenda includes the performance of RMI or "remote method invocation" which is the attractive Java distributed object model. More broadly the concern is the performance of Java runtime (the virtual machine VM) with lots of threads, I/O, and large memory use. There are several important parallel computing interfaces including Java MPI binding and higher level interfaces such as that in the data parallel HPJava mentioned in the final section 5.

The Grande organization effort is working on two MPI bindings for Java. One is a natural extension of current C++ binding. A second more powerful approach ignores MPI's Fortran and C heritage and exploits the object structure of Java. The current proposals for Java MPI bindings are available at the Java Grande homepage and are based on three current implementations. Firstly, *mpiJava* from NPAC is modelled after the C++ binding for MPI and implemented through JNI wrappers to native MPI software.[21] Secondly, *JavaMPI* from Westminster University London uses automatic generation of wrappers to legacy MPI libraries using a C as opposed to C++-like implementation.[22] Lastly, *MPIJ* is a pure Java implementation of MPI from BYU, which is closely based on the C++ binding. A large subset of MPI has been implemented using native marshaling. [23]

As well as standardizing these interfaces, the group is developing a framework for a universal Java Seamless interface to computing resources, which we will discuss in the next section.[24] Finally a major activity is the development of Grande Application benchmarks. This overlaps the activities of the first working group, which has already started an interesting numeric kernel collection at NIST[10]based on the ideas pioneered in the Java version of LinPack [25]. The Edinburgh group has started a collection while is online at [26].

Both working groups have made substantial progress in the last few months with initial reports including key issues we need to bring up with Sun in both the *Numerics* and RMI performance areas. We need broad community involvement in critiquing our proposals, collecting Java Grande benchmarks, and defining standard classes and libraries. We hope to get good participation in a set of workshops on the seamless computing framework. We also need applications that will stress Java and Java runtime (the VM) with large applications – we need to find those weak links of the VM, which lead to performance problems? Note that enterprise Intranets will lead to some such scaling tests but there are some features that will only appear with Grande problems.

4 Desktop Access to Remote Resources

Many computer users are not so interested in the full features of metacomputing but rather in being able to run their jobs in a seamless way that does not keep changing as backend computer resources are upgraded. Viewing computing as a distributed (object) service, we can satisfy this need by developing a "Java Framework for Computing Services". This enables development of web interfaces to run a given job on any computer with any data source compliant with this framework. This is analogous to JDBC (Java Database Connectivity), which gives a universal interface to any relational database. Our proposed framework can be used in metacomputing environments to allow linkage of multiple computers to run together on a single job. Such a framework can be implemented in other approaches and form for instance, a CORBA vertical facility. In fact, we are defining methods and properties of computers and programs viewed as distributed objects. We term the effort generically as DATORR, which stands for Desktop Access to Remote Resources. The Computing Services Framework will allow vendors to compete on either the user front-end (GUI) or back end services with the DATORR framework providing universal linkage.[27,28] We brought together many of the major metacomputing [17-20] and universal interface[29,30] project leaders together for a meeting on Oct 8-9,98 at Argonne and held a successful SC98 Birds of a Feather session. We are collecting information on projects and abstracting requirements from user and system point of view. We hope over the next year to build consensus on standard interfaces and so develop the proposed Java framework for computing services. Information on this effort can be found at [24] with email reflector datorr@mcs.anl.gov.

We now list some of the possible services whose interfaces could be defined in the DATORR framework. There is the area of Grande resource discovery, allocation and scheduling, where the recent JINI Sun technology looks attractive. Other services could include compiling and executing jobs with a specification of features needed for execution optimization. The latter includes parameters needed by MPI/HPF decompositions such as the number of processors. A key capability would be interfaces to resource management and scheduling systems such as Codine or LSF.[31,32] More mundanely but perhaps more importantly, we need universal interfaces to accounting and here we may be able to take advantage of Web electronic commerce technology. Security and authentication services are essential and especially hard in metacomputing where one must link several different management policies. Again, the public key infrastructure being developed for Internet commerce will be very important.

5 Parallelism in Java

In order to discuss parallelism in Java, we divide the forms of parallelism seen in applications into four broad categories.

1. **Modest Grain Size Functional Parallelism:** Here we are thinking of the type of parallelism used when computation and I/O operation are overlapped as exploited extensively by web browsers but can also be seen in possible overlap of message traffic with computation in the Java plus MPI parallel programming paradigm. This parallelism is built into the Java language with threads but has to be added explicitly with (thread) libraries for Fortran and C++. In the browsers, one typically sees the different components (multiple images, applets etc.) processed by different threads concurrently.

2. **Object Parallelism:** This is quite natural for C++ or Java where the latter can use the applet mechanism to portably represent and invoke concurrently objects. This familiar in distributed military simulations that already use large-scale object based models including a standard HLA (High Level Architecture) distributed object framework.[33] We have in fact successfully implemented a Java Server based approach to this, called WebHLA and believe that base Java capabilities are well suited to this field.[5-8]

3. **Metaproblems:** This is the parallelism in applications that are made up of several different sub-problems which themselves may be sequential or parallel. Multidisciplinary applications such as the linkage of ocean-atmosphere or fluid flow-structures illustrate this problem class. The middle tier shown in figure 1 is precisely a metaproblem.

4. **Data Parallelism:** Here we refer to natural large-scale parallelism found from parallel updates of grid-points, particles and other basic components in scientific computations. Such parallelism is supported in Fortran (or C) 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 than in C or C++. 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++). An effort at NPAC [34] focuses on this approach but uses a high level SPMD rather than the traditional HPF style approach to data parallel Java. We should note that the powerful standard template library approach used in C++ by POOMA[35] for data parallelism cannot be supported in Java. The implications of this are unclear at present.

Data parallelism can be supported using threads on shared memory machines as pioneered at Indiana [36] whereas in distributed memory machines, explicit message passing must be used at the low level. This leads to the hybrid model of fig 2. One advantage of distributed shared memory architectures is their support of a single Java VM and a uniform programming model. Comparing with the discussion of category 2 above, we see that threads can of course be used to support multiple forms of relatively fine grain parallelism. Message passing is clearly satisfactory for Java as the language naturally supports inter-program communication, and as described in sec. 3, the standard capabilities of high-performance message passing (MPI) are being implemented for Java.

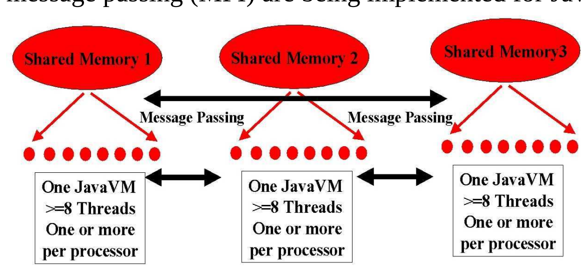

Figure 2: *Hybrid Java Model for Parallelism*

In summary, Java directly addresses three of the four forms of parallelism described above. In these areas, it seems equal to or superior to other languages. Java needs to be augmented to fully support data parallelism but so do Fortran and C++.

References

[1] Workshop on the Use of Java in Science and Engineering Computation, Syracuse December 16-17,98; Proceeding edited by Geoffrey Fox and published in Concurrency: Practice and Experience Vol 9 Issue 6 June 1997. Online at http://www.npac.syr.edu/projects/javaforcse/javameettalks.html

[2] ACM 1997 Workshop on Java for Science and Engineering Computation June 21 1997 held as part of PPoPP in Las Vegas; Proceedings edited by Geoffrey Fox and Wei Li and published in Concurrency: Practice and Experience Vol 9 Issue 11 November 1997. Online at http://www.cs.ucsb.edu/conferences/java98/

[3] ACM 1998 Workshop on Java for High-Performance Network Computing at Palo Alto Feb 28-March 1 1998; Proceedings edited by Siamak Hassanzadeh and Klaus Schauser published in Concurrency: Practice and Experience, Vol 10 Issue 11 November 1998. Online at http://www.cs.ucsb.edu/conferences/java98/

[4] First European Workshop on Java for High-Performance Network Computing at Southampton as part of Europar '98, September 2-3 98. Online at http://www.cs.cf.ac.uk/hpjworkshop/

[5] G. C. Fox, W. Furmanski and H. T. Ozdemir, Java/CORBA based Real-Time Infrastructure to Integrate Event-Driven Simulations, Collaboration and Distributed Object / Componentware Computing, In Proceedings of Parallel and Distributed Processing Technologies and Applications PDPTA '98, Las Vegas, Nevada, July 13-16, 1998, online at http://tapetus.npac.syr.edu/iwt98/pm/documents/

[6] Geoffrey Fox and Wojtek Furmanski, "Petaops and Exaops: Supercomputing on the Web", IEEE Internet Computing, 1(2), 38-46 (1997); http://www.npac.syr.edu/users/gcf/petastuff/petaweb

[7] Geoffrey Fox and Wojtek Furmanski, "Java for Parallel Computing and as a General Language for Scientific and Engineering Simulation and Modeling", Concurrency: Practice and Experience 9(6), 4135-426(1997).

[8] Geoffrey Fox, Wojtek Furmanski, Hasan T. Ozdemir and Shrideep Pallickara, "High Performance Commodity Computing on the Pragmatic Object Web", 1998 unpublished

[9] Resource on Java Numerics maintained by Ron Boisvert and Roldan Pozo at http://math.nist.gov/javanumerics/

[10] Java Numerical Benchmarks maintained by Roldan Pozo at http://math.nist.gov/scimark/

[11] NPAC's Java Academy for Middle and High School Students produced by Tom Scavo Spring 1998 online at http://www.npac.syr.edu/projects/k12javaspring98/

[12] The Java Grande Forum with Home Page at http://www.javagrande.org/

[13] NPAC Resource for the use of Java in Computational Science and Engineering online at http://www.npac.syr.edu/Java/index.html

[14] Floating Point Performance in Java presented at a Java Grande Forum Meeting May 9 1998 by Marc Snir (from work with Jose Moreira, Manish Gupta, Lois Haibt and Sam Midkiff) and online at http://www.javagrande.org/ibmgrande.pdf

[15] Optimizing Java -- Theory and Practice by Zoran Budimlic and Ken Kennedy online as in Ref. [1] and published Concurrency: Practice and Experience 9,445-464(97)

[16] Z. Budimlic, K. Kennedy and J. Piper The Cost of Being Object-Oriented: A Preliminary Study online in Ref. [4]

[17] The Grid: Blueprint for a New Computing Infrastructure Edited by Ian Foster and Carl Kesselman, Morgan Kaufmann Publishers, San Francisco, California 1998

[18] LEGION, HPCC metacomputing environment developed by Andrew Grimshaw *http://www.cs.virginia.edu/~legion/*

[19] E. Akarsu, G. Fox, W. Furmanski and T. Haupt, "WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", paper submitted for Supercomputing 98, http://www.npac.syr.edu/users/haupt/ALLIANCE/sc98.html

[20] I. Foster and C. Kessleman, Globus Metacomputing Toolkit, http://www.globus.org

[21] *mpiJava* binding of Java for MPI from NPAC at Syracuse University http://www.npac.syr.edu/projects/pcrc/HPJava/

[22] *JavaMPI* binding of Java for MPI from Westminster University London http://perun.hscs.wmin.ac.uk/JavaMPI/

[23] *MPIJ* binding of Java for MPI from Brigham Young University. http://ccc.cs.byu.edu/DOGMA/

[24] DATORR DeskTop Access to Remote Resources Home Page at http://www-fp.mcs.anl.gov/~gregor/datorr

[25] Linpack Benchmark -- Java Version by Jack Dongarra online at http://www.netlib.org/benchmark/linpackjava/

[26] Benchmark collection from Edinburgh parallel computer center. http://www.epcc.ed.ac.uk/research/javagrande/benchmarking.html

[27] A seamless computing technology resource maintained by Mark Baker online at http://www.sis.port.ac.uk/~mab/Computing-FrameWork/

[28] A list of useful seamless computing links is maintained in Japan by the Real World Computing Partnership and is online at http://www.rwcp.or.jp/people/ishikawa/seamless.html

[29] WebSubmit web interface to HPCC resources from NIST online at http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html

[30] Unicore: Uniform Interface to Computing Resources project in Germany led by Dietmar Erwin and online at http://www.kfa-juelich.de/zam/RD/coop/unicore/

[31] Codine commercial Resource Management System from Genias online at http://www.genias.de/products/codine/description.html

[32] Load Sharing Facility LSF commercial Resource Management System from Platform Computing online at http://www.platform.com/

[33] High Level Architecture and Run-Time Infrastructure by DoD Modeling and Simulation Office (DMSO), online at http://www.dmso.mil/hla

[34] Guansong Zhang, Bryan Carpenter, Geoffrey Fox, Xinying Li, and Yuhong Wen, Considerations in HPJava language design and implementation. In 11th International Workshop on Languages and Compilers for Parallel Computing, North Carolina August 1998. Online at http://www.npac.syr.edu/projects/pcrc/HPJava/

[35] POOMA, Los Alamos Object Oriented Scientific Computing Environment led by John Reynders, http://www.acl.lanl.gov/pooma/main.html

[36] Javar: A prototype Java Restructuring Compiler by A.J. Bik, J.E. Villacis and D.B. Gannon, Concurrency: Practice and Experience 9,1181-1192(97)