Java Access to Numerical Libraries

Henri Casanova * Jack Dongarra* † David M. Doolin*

May 31, 1997

Abstract

It is a common and somewhat erroneous belief that Java will always be "too slow" for scientific computing. Two projects underway at the University of Tennessee are addressing the question of scientific computing via Java: *NetSolve* and *f2j*. The approaches taken by these two projects are radically different. NetSolve allows users to access pre-installed computational resources, such as hardware and software, distributed across the network. Using these resources, the user can easily perform scientific computing tasks without having any computing resource installed on his or her computer. NetSolve features a Graphical User Interface written in Java as well as a Java Application Programming Interface. The f2j (Fortran to Java) project will provide the numerical subroutines translated from their Fortran source into class files suitable for use by Java programmers. This makes it possible for a Java application or applet to use established legacy numerical code that was originally written in Fortran. This article describes the research issues involved in these two projects and their current limitations. We also explains how, although using two different paradigms and addressing somewhat different classes of users and applications, NetSolve and f2j achieve a common goal: to provide efficient, reliable and portable access to standard numerical libraries via Java.

Keywords

Java, bytecode, Assembly, Jasmin, Scientific Computing, Numerical Libraries
Computational Servers, Internet-computing, Compilers, Software Repositories

*Department of Computer Science, University of Tennessee, TN 37996
†Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

1 Introduction

The Java language has become very successful since its formal introduction in 1995. The language itself is very attractive: it is object oriented and standard class libraries provide very natural interfaces to features such as multi-threading, Internet communication and protocols, graphical components, Graphical User Interface (GUI) design facilities, customizable security restrictions, etc. More than the language itself, Java's popularity may be attributed in part by its ability to be executed as *applets* running within Web browsers. Java source code is compiled into hardware independent *bytecode,* which is then interpreted by a Java Virtual Machine (JVM). A JVM can be embedded in a Web Browser and, with the appropriate security restrictions, execute bytecode programs called class files that are accessible from the World Wide Web. JVMs are being provided by an increasing number of software developers, sometimes as a standard feature of their operating systems. It is possible to run Java *applications* on these architectures as executable programs. To increase the performance of Java, just-in-time (JIT) and native code compilers have also been developed. JIT compilers translate Java byte code to native code at run time. Native code compilers produce machine specific executables from Java source code. The increase in execution speed is balanced in each case by a corresponding decrease in platform independence.

Faced with the increasing acceptance of Java as a viable programming language and the fact that it can be used for multi-threaded Internet-based distributed applications, it is natural to consider using Java for scientific computing. Scientific computing applications are generally written using pre-existing libraries that address different fields of computational science (linear algebra, optimization, curve fitting, etc.). It would, of course, be possible to hand-write directly, in Java, the numerical class libraries needed by Java programmers, but, considering the number of numerical libraries available, the number of functions in each of these libraries and the number of years that were invested in their development, rewriting and testing all this software in Java would be a formidable task.

Several solutions can be found to avoid such a rewrite while still providing Java programs with access to the numerical algorithms already implemented in Fortran. Two projects currently being developed at the University of Tennessee. One approach is to allow Java applications or applets to access computational servers that can directly use pre-installed Fortran libraries. Such a scheme is enabled by the NetSolve project which is the object of section 2. As detailed in that section, NetSolve provides in fact much more than just this computational server paradigm, and Java is only one of the ways to access the NetSolve resources. The other approach is to translate the Fortran source code of the libraries of interest into Java class files. The f2j project is a first attempt at such translation and is described in section 3.

2 Access to numerical software via NetSolve

2.1 Rationale

Scientific computing has been a major part of both research centers and industry for many years. As such, it has been the object of many investigations which have led to the development of numerous software products. These products can be classified into different categories. Some numerical tools, like MATLAB [1] or Mathematica [2], have enjoyed great success. These tools generally provide an interactive interface as well as the possibility of writing scripts to perform computation.

Another class of products falls under the category of numerical libraries. Numerical libraries are less convenient than interactive tools, since the user generally is required to write a C or Fortran program. Nevertheless, they offer the advantages of execution speed and flexibility. A very large number of such libraries exists and they cover diverse fields of computational science. Moreover, unlike interactive tools, numerical libraries are often freely available (e.g., LAPACK).

A third class of tools comprises runtime packages whose goal is to help the user perform some specific type of built-in, scientific computation. Like numerical libraries, such packages are usually freely available. One example is NEOS [3], which is focused on linear programming and optimization.

Users wanting to solve a numerical problem are thus confronted with a dilemma. They can purchase a commercial product and take the risk that it might not be suitable for future use on different kinds of problems, or they can try to locate and download free libraries and write programs in terms of specific functions or subroutines. NetSolve addresses this second situation, where the user is confronted with many difficulties.

The first task the user must undertake is to look for the appropriate library or set of libraries he or she needs for their own computational problem. Usually, such libraries can be found in software repositories. One well-known repository, for example, is Netlib [4], which is maintained through the collaborative effort of several institutions and universities. Software repositories present some intrinsic difficulties for the inexperienced user: (i) they are generally very large and (ii) contain very different types of libraries. Once located, the appropriate library must be downloaded and installed. Depending on the nature of the software, this step might be nontrivial, especially for a user having no experience with this kind of task. However, the biggest step still remains; learning how to use the library itself, that is, how to write a program in terms of its library components. Such a task can be formidable and time-consuming (even without regard for the debugging phase). Moreover, the user may want to write the program or script in a language that is different from the language the numerical library has been written in. Even inter-language interoperability is possible, the user must learn the techniques that are often non-portable and require experience.

These considerations motivated the establishment of the NetSolve project. NetSolve is a client-server application designed to solve computational science problems over a network. A number of different interfaces have been developed for the NetSolve software so that users of C, Fortran, MATLAB, Java or the Web can easily use the NetSolve system. The underlying computational software can be any scientific package, thereby ensuring good performance results. Moreover, NetSolve uses a load-balancing strategy to improve the use of the computational resources available. The following section gives an overview of the NetSolve system.

2.2 Overview of the NetSolve project

2.2.1 Architecture

The structure of the NetSolve system organization is depicted in Figure 1. To solve the challenges highlighted above, NetSolve provides the user with a pool of **computational resources**. These resources are in fact computational servers that provide run-time access to arbitrarily specified numerical libraries. The user can use one of the different NetSolve **client** interfaces to send requests to these servers. The user requests, however, are not sent directly to the computational resources, but are instead processed by another component of the system: a NetSolve **agent**. The agent decides which computational server should handle the user request and assigns the request to that server.

The different hosts that participate in the NetSolve protocol may be located anywhere on the Internet. In fact, they may be administered by different institutions. NetSolve does not assume any centralized control over the different hosts in the system. On the contrary, each process (computational server or agent) is an independent entity: it can be stopped/restarted safely at anytime without putting the integrity of the system in jeopardy. Furthermore, a NetSolve system can contain several instances of the NetSolve agent. Suppose, for example, the set of computational resources spans several local area networks and that users on each of these networks want to use NetSolve to perform scientific computations. It is then possible to start a NetSolve agent on each network, so that user requests always go to the "closest" agent to be processed. Different instances of the NetSolve agent can then have different views of the set of computational resources, reflecting the fact that certain clients are closer to certain computational resources.

The specification of the computational servers and how they interact with the underlying numerical libraries is detailed in [5] where it is explained that computational servers are created with the help of a Java applet. Currently, computational servers interfacing with the following numerical libraries have been successfully created: FitPack [6], ItPack [7], MinPack [8], FFTPACK [9], LAPACK [10], BLAS [11, 12, 13], and QMR [14].

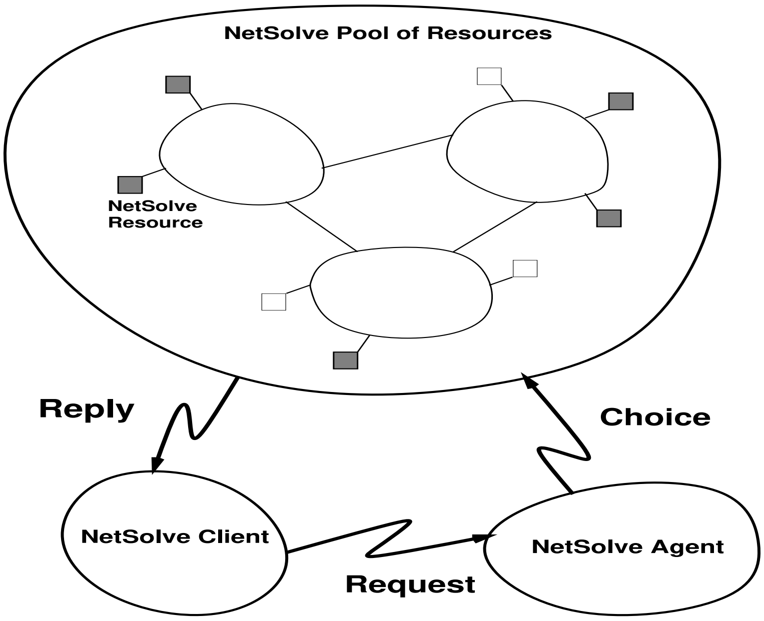

Figure 1: The NetSolve System

2.2.2 Load Balancing

The primary role of the agent in the NetSolve system is to perform load balancing among the different computational resources. NetSolve is inherently a multi-request system. Several users can share computational resources by contacting the same agent or different agents managing the same pool of resources. In fact, even a single user can send multiple asynchronous requests at once, as explained in the description of the user interfaces. For each server, the agent uses information contained in the user request (e.g., type of computation, size of the problem), static information about the server (speed of the host, numerical server available, etc.), and **predictions** about the workload of the server's host and the distance to the server's host over the network. These different pieces of information are then combined to obtain an estimate of the time required to process the user request on each computational server (including network time and CPU time). For each client incoming request, the NetSolve agent sorts the appropriate computational servers by the estimated times, and is then able to process the request accordingly. More details on the way the agent performs this load balancing can be found in [15].

2.2.3 Fault Tolerance

As previously mentioned, the hosts in the NetSolve system can be located anywhere on the Internet and can therefore be administered by different institutions. For this reason, NetSolve does not try to impose any control on the different resources. This distributed computation approach is very flexible, but it requires that NetSolve implement fault tolerance mechanisms. Indeed, any resource can become unreachable at any moment, perhaps because of a network failure, a host failure, or simply because a system administrator reboots a host.

The NetSolve system ensures that a user request will be completed unless every single resource has failed. When a client sends a request to a NetSolve agent, it receives a sorted list of computational servers to try. When one of these servers has been successfully contacted, the numerical computation is started. If the contacted server fails during the computation, then another server is contacted and the computation is restarted. This whole process is transparent to the user. If all the servers have failed, then the user is notified that the computation can not be performed at that time. The server-list strategy represents a first step towards fault tolerance and will be improved in future versions of the software.

2.2.4 Multiple User Interfaces

An important goal of the NetSolve design is to provide several interfaces for a wide range of target users. Presently, NetSolve provides C, Fortran, MATLAB and Java Application Programming interfaces (APIs). We also deemed that NetSolve needed a graphical interface and this interface has been written in Java as well. The two Java interfaces are described at the end of this section.

Another concern is to keep the interface as simple a possible. For example, the MATLAB interface contains only 2 functions : `netsolve()` and `netsolve_nb()`. The first function allows the user to send *blocking* requests to NetSolve whereas the second one sends *non-blocking* requests. In fact, every interface provides non-blocking calls to NetSolve. When several non-blocking requests are sent to a NetSolve agent, they are dispatched between the available computational resources according to the load balancing schemes implemented by the agent. With minimal effort, the NetSolve user can achieve a degree of parallelism. Complete details regarding the various interfaces as well as examples can be found in [16].

2.3 Java interface to NetSolve

2.3.1 Graphical interface

For NetSolve to be accepted by many users, it must provide several distinct ways to access computational resources, mainly different user interfaces. Before the development of the Java graphical interface, the only interfaces that were available were MATLAB, C, and Fortran, leaving the MATLAB interface as the only high-level and interactive way of calling NetSolve. However, MATLAB is not freely distributed and many users may not have it installed on their workstations. We needed a freely available interactive interface so that users not familiar with scientific computing could still use NetSolve for their first steps in the field. Furthermore, Java allowed us to effortlessly design a graphical interface making NetSolve even more straightforward for first-time users, even though an experienced user is more likely to use the Fortran or MATLAB interface if available.

NetSolve's interactive Java interface is entirely graphical. A window displaying available computational methods is presented to the user, who can point and click to solve the numerical problem of interest. The user then supplies input by way of files, URLs, or interactively using the keyboard. Computation is handled as usual by a computational server chosen by the NetSolve agent and the result is returned to the user. The results can be viewed and saved if necessary.

NetSolve's graphical interface is multi-threaded, using Java's built-in thread facilities. Each computation is assigned to a separate thread. Every request sent by the Java interface to the NetSolve computational resource is represented by an independent window. By opening several windows concurrently, the user can send multiple requests simultaneously. Each request will be handled by different computational servers, thanks to the agent's load balancing strategy. This is very similar to the non-blocking calls to NetSolve from C, Fortran, MATLAB or Java programs.

The pool of NetSolve resources can be a heterogeneous set of hosts, i.e., two hosts can have different internal data representations. To support heterogeneous environments, the agents, servers, and C, Fortran and MATLAB interfaces use the XDR protocol [17]. As part of NetSolve, we developed a simple Java XDR-encoder, enabling our interface to communicate with any server on any platform. The Java interface may then contact servers which expect XDR-encoded data.

One of the features of Java is that the Java virtual machine can enforce a strict security scheme. Typically, the virtual machine embedded in a Web-browser does not allow local file access, network connection except to the Web server that served the applet. Presently, the Java interface to NetSolve cannot be used as an applet within a Web browser because of these security restrictions. Indeed, our interface is supposed to contact many different servers scattered on the Internet and therefore perform multiple network connections.

It is possible to work around applet security restrictions by setting up an appropriate daemon on the Web servers, but this would not be a viable solution in the long term, considering the mass of data that would transit through the Web server. However, the interface is fully functional as an application, running on a system which has a Java virtual machine, or distributed with a Java Runtime Environment. Considering the increased interest in Java, most systems already have an implementation of the Java Development Kit (JDK) that includes the Java virtual machine and the javac compiler. Our interface should therefore be usable on virtually any platform and future versions of Web-browsers allowing security levels to be set by the user are likely to make the applet approach feasible.

2.3.2 NetSolve's Java API

The first graphical interface to NetSolve was written in TK/TCL [18]. When Java became available, we decided that it would be a much better choice to write a graphical interface for the reasons described in the preceding section. By the time the first version of the Java GUI was completed, Java had evolved to be accepted as an important language by itself, not just a tool for running programs (applets) within Web-browsers. The need for a Java API to NetSolve then seemed clear, since NetSolve already provided C, Fortran and MATLAB interfaces. Fortunately, most of the internals of the GUI could be re-used to implement a Java API. In fact, due to the unexpected success of Java during the development of NetSolve, we were lead to do some reverse engineering: extracting the internals of an existing software for distribution as a stand-alone class library. The API is still under development and has motivated some code restructuring within the GUI. It will match the other APIs to NetSolve as closely as possible.

There are however some difficulties that are inherent to Java. For instance, Java does not support calling sequences to a same function with a variable number of arguments. Also, as explained in section 3, Java passes primitive types by value which does not allow a calling sequence entirely compatible with the underlying Fortran numerical libraries in the NetSolve computational servers. These difficulties are, however, surmountable, and an attractive API will be presented for providing users with an ability to write portable scientific computation applications and applets. The user developed programs will be reliable because the NetSolve computational servers use established and widely accepted numerical libraries. They will also be efficient because of NetSolve's agent-based design, allowing the user to achieve parallelism with virtually no additional programming effort.

2.4 Future Developments

NetSolve is still a relatively young, rapidly developing project which is bound to greatly improve in the near future. The Java GUI and API to NetSolve will reflect these improvements when appropriate, evolving to provide the user with attractive new features. The most interesting improvements will most likely take place within the Java GUI since the API should stay as simple as possible. The GUI on the other hand can integrate new modules on demand. For instance, it would be very feasible to add data visualization modules to the Java GUI so that the user can be presented with the results of his or her computations in an attractive graphical display. Another possible extension would be to develop some kind of MATLAB-like scripting language with which the user could describe a whole numerical algorithm using the NetSolve services. In fact, better than a scripting language, would be a graphical interface describing a numerical algorithm as a task-graph and where each node would be an invocation to a NetSolve service. We have already started developing a prototype of such a graphical interface in Java that could be integrated in the NetSolve Java GUI. This prototype could eventually fit in an even broader framework where the nodes of the graph are not limited to a NetSolve service but perhaps describe a more general task, like a user's Java program. Following this thread of ideas and new developments in the Java world, it appears that users might want to call numerical software *directly* from their applications or applets, instead of using NetSolve. This might indeed be a good choice for users concerned with security and who do not want their data to travel over the Internet, or users whose applications do not require a large computational power, or for users who have access to hardware computational resources and plan to use some kind of native Java compiler, to ensure acceptable performance levels. The following section describes a new project, just started at the University of Tennessee, that will allow such users to include calls to numerical libraries in their Java programs.

3 Direct Java access to numerical software

3.1 Motivation

> *Real programmers program in Fortran, and can do so in any language. —Ian Graham, 1994 [19]*

Following the development of NetSolve, the f2j project, recently started at the University of Tennessee, will provide APIs for direct access to numerical libraries from Java programs. f2j is a formal compiler that translates programs written using a subset of Fortran 77 into a form that may be compiled or assembled into Java class files. The first priority for f2j is to translate the BLAS [11, 12, 13] and LAPACK [10] numerical libraries from their Fortran 77 source code to Java class files. The subset of Fortran 77 translated by f2j matches the Fortran source used by BLAS and LAPACK. These libraries are established, reliable and widely accepted linear algebra packages, and are therefore a reasonable first testbed for our translator. Many other libraries of interest are expected to use a very similar subset of Fortran 77.

The primary motivation for this project is to provide the reliability and dependability of the LAPACK numerical linear algebra subroutines as class files available for use in the Java Virtual Machine (JVM). Targeting the JVM provides access to legacy code for distributed computation via the World Wide Web. Similar previous efforts such as f2c [20], have proven to be very popular and widely used. The BLAS and LAPACK class files will be provided as a service of the Netlib repository. f2j also provides a base for a more ambitious effort translating a larger subset of Fortran, and perhaps eventually *any* Fortran source into Java class files.

Popular opinion seems to hold the somewhat erroneous view that Java is "too slow" for numerical programming. However, there are presently many small to intermediate scale problems where speed is not an issue. For instance, physical quantities such as permeability, stress and strain are commonly represented by ellipsoids [21, 22], a graphical representation of an underlying tensor. The tensor is mathematically represented by an SPD matrix. Ellipsoid axes are computed from the root inverse of the tensor's eigenvalues, directed along the tensor's eigenvectors. A LAPACK eigenproblem subroutine such as SSYTRD, available as a Java class file, provides a portable solution with known reliability. Since future execution speeds of Java will increase as just-in-time (JIT) and native code compilers are developed, the scale of feasible numerical programming will increase as well.

3.2 Implementation

Several freely available Fortran compiler fronts ends, such as g77 and f2c, were examined to base the f2j on. None of these fit the needs of the project sufficiently well, and the following quotation helped provide motivation to start a clean implementation from scratch:

> *The program f2c is a horror, based on ancient code and hacked unmercifully. Users are only supposed to look at its C output, not at its appalling inner workings. —Stuart Feldman [20]*

Due to the context sensitive nature of the Fortran language, the lexer was hand written (in C), as recommended in [23, 24]:

> *It should be noted that tokenizing Fortran is such an irregular task that it is frequently easier to write an ad hoc lexical analyzer for Fortran in a conventional programming language than it is to use an automatic lexical analyzer generator. —Alfred Aho, 1988 [23]*

This allowed expression of the Fortran grammar as LR(1), sufficient to use the parser generator Bison, a yacc work-alike distributed by the Free Software Foundation. Bison generates a an ANSI C parser, which is useful for platform indendence. There is no type checking if Java source code is chosen to be emitted. The Fortran source file is assumed to be standard Fortran 77. Limited type checking is done during the type conversion pass when assembler opcode is emitted. All code generation and type conversion procedures are written in C, portability and extensibility.

The Fortran source code is parsed into an abstract syntax tree (AST) consisting of tagged union nodes implementing the equivalent Java structures. Using an abstract syntax tree has several benefits, one of which is that code restructuring can be easily performed. For instance, continuing loop iteration within a Fortran do loop requires a `goto/continue` pair, in Java this is accomplished similar to C, with a `continue` statement. Similarly, breaking a loop in Fortran 77 requires a `goto/label` pair, implemented as a `break` statement in Java. The AST allows easy lookup and connection between non-adjacent nodes for such code restructuring. Another benefit is that the AST may be passed by its root node to separate type-checking, code optimizing and code generation procedures.

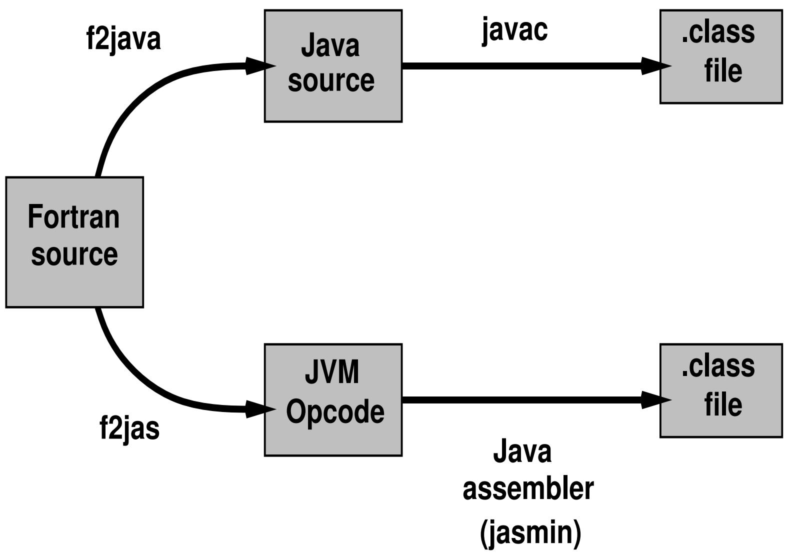

Figure 2: Translation strategies in the f2j project

After parsing a Fortran source file, the AST is traversed recursively to emit either Java source code for compilation or Java opcode suitable for assembly into class file format. Targeting Java source and opcode is more convenient than producing bytecode directly because: (i) internal documentation of BLAS and LAPACK subroutines exists in the form of comment headers and can be preserved exactly in the translated form, and (ii) Java source and opcode are stored in readable ASCII text files, much more convenient for testing and debugging the translated routines. Targeting Java source is fairly straightforward, but due to many control structures in the BLAS and LAPACK reference source being written with goto constructions, the amount of Java source code presently emitted is limited. Since code restructuring has been shelved for later consideration, the remaining subroutines are emitted as opcode suitable for assembly using a public domain assembler, *jasmin* (Java Assembler Interface) [25]. JVM opcode has not yet been standardized by Sun, but jasmin uses instructions identical to those specified by Sun in their JVM documentation [26]. The same opcode is produced as output from Sun's javap program invoked to disassemble a Java class file. Figure 2 shows a diagram of the two translation schemes that are being experimented with in the f2j project.

One of the more challenging aspects of the project is resolving differences between the calling structures in Fortran and Java. Fortran passes all arguments by reference. Java passes objects by reference, but primitives such as integers, floats and doubles are passed by value. A Fortran subroutine that modifies an integer for use in the calling program has no direct counterpart in Java. One possibility would be to create a class containing all of the arguments for a subroutine. While this technique would be more object oriented than changing lists of calling parameters, the complexity of writing a translator could be greatly increased.

Java provides class wrappers for all primitive types, but value of the primitive is immutable within the object wrapper. Since the JVM requires that all methods return a value or void, values cannot be left on the JVM stack (each method implements its own stack). The solution being implemented is to simply wrap the necessary values (e.g., the INFO variable used in many LAPACK routines for reporting error status) in a custom class consisting of a single static class variable. This also provides less overhead than instantiating built-in wrappers provided in the Java language specification. A simple experiment showed that instantiating an object of type Double requires 280 bytes in Java (javac version 1.1.1), but a simple wrapper such as

```
class DoubleWrapper {
  double d;
}
```

only requires 56 bytes. Both classes, Double and DoubleWrapper, inherit from class Object. The size difference reflects methods implemented in Double that are lacking in DoubleWrapper. Another possibility under consideration is to pass primitives as single element arrays.

Neither is it possible to pass references to subsections of arrays. Java will dereference indexed arrays and pass the value instead. This necessitates changing the calling parameters of the BLAS and LAPACK routines to pass indices separately with every array. Method overloading would allow a default method invocation identical to a LAPACK call that passed all arrays by their initial reference, but overloading for all possible cases would increase the number of required methods by 2^n, n the number of arrays.

3.3 Future Developments

As the initial design is very much a proof of concept, the compiler is limited to the double precision BLAS and LAPACK routines. Future work on the project might include providing better type checking, or providing BLAS and LAPACK in the form of Java language source. Providing Java language source would require formal code restructuring to translate Fortran goto statements to equivalent Java control structures, as discussed in section 3.2. Since the present compiler produces assembler code directly, to handle goto statements, it could be modified to produce stack values suitable for complex arithmetic.

Currently, implementing complex numbers and complex arithmetic would raise performance issues. Complex numbers are not specified as primitives in Java, and while they could be easily implemented as objects, this would produce considerable overhead, as demonstrated above. A better idea would be to define static methods to operate directly on a stack containing the appropriate arguments for complex number arithmetic. This solution still suffers a performance penalty with each static method invocation. The best solution would of course be complex primitives specified by Sun as part of the Java language, with associated JVM instructions to provide complex arithmetic capability.

Another extension of f2j could provide simple code optimization. Direct translation of the Fortran statement `i = i + 1` to increment an index `i` results in the following JVM stack sequence:

```
iload n           ; Push n to stack.
iconst_1          ; Push integer constant 1 to stack.
iadd              ; Integer add.
istore n          ; Store value on stack in n.
```

where `n` is a variable local to the methods stack, and `;` delimits comments. Optimizing this operation using JVM instructions results in

```
iinc n 1    ; Increment the value stored in n by 1.
```

For large or highly iterative procedures, the savings in the size of byte code and execution speed should be significant.

As well as providing numerical routines for Java programming, the Fortran to Java compiler f2j demonstrates the feasibility of compiling arbitrary languages to the JVM using the class file format. Fortran to Java source translations may also be accomplished when control structures in Fortran match those in Java, or can be restructured to match. This effort certainly will not be the first such, indeed, a C++ to Java translator already exists, and work is underway on a Pascal to class file translator (Mark de Boer, personal communication).

4 Conclusion

As Java's popularity grows, it becomes more and more crucial to provide reliable ways to write Java programs that perform scientific computation. There are basically two reasons why Java is often claimed to be inappropriate for scientific computing. The first reason is that Java is generally interpreted and is therefore unable to run at machine speed. The second reason is that no standard numerical libraries have been translated yet to Java bytecode (or source for that matter). The first reason seems to be less and less or a concern as native compilers are being provided by software developers, so that Java can be used as any other language to develop code on those platforms. Even the speed of applets can be raised to new levels with JIT techniques. However, compiled Java source on those platforms is of course not portable anymore as it is not targeting the JVM any more.

Two projects underway at the University of Tennessee will provide efficient, portable and reliable access to scientific computing facilities using Java: NetSolve and f2j. Both projects achieve reliability by providing access to standard numerical libraries. NetSolve possesses a framework that allow easy integration of arbitrary numerical libraries hosted on remote computational servers. f2j is for now limited to libraries using a subset of Fortran 77. This subset is used by the BLAS and LAPACK libraries and is likely to be also sufficient for other standard numerical libraries or at least easily expendable. Portability is also achieved by both NetSolve and f2j. NetSolve's Java API of course leads to portable code since it is at the Java source level. f2j takes in input Fortran source and ultimately generates portable Java class files. Two ways of generating this bytecode are under investigation: either compiling Java source or assembling JVM opcode. Going through Java source would be of course extremely interesting, but as explained in section 3, it will require code restructuring. Going through Java opcode is somewhat easier but is then bound to a JVM execution. Finally, these two projects are also concerned with efficiency. NetSolve insures efficiency thanks to an agent-based design and computational servers that run optimized compiled Fortran numerical library functions. f2j depends on the efficiency of the run-time environment in which the generated class files are executed. Future work with f2j will minimize overhead by optimizing the emitted JVM opcode.

The NetSolve and f2j are emerging developmental projects and are quite likely to undergo substantial modification and improvement as the involved research issues become better understood. Both projects aim at making Java-based scientific computing not only feasible but attractive for a broad class of applications and users.

References

[1] Inc. The Math Works. *MATLAB Reference Guide*. 1992.

[2] S. Wolfram. *The Mathematica Book, Third Edition*. Wolfram Median, Inc. and Cambridge University Press, 1996.

[3] J. Czyzyk, M. Mesnier, and J. Moré. Neos : The Network-Enabled Optimization System. Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[4] S. Browne, J. Dongarra, E. Grosse, and T. Rowan. The Netlib Mathematical Software Repository. *D-Lib Magazine*, sep 1995. Accessible at http://www.dlib.org/.

[5] H Casanova and J. Dongarra. The use of Java in the NetSolve project Computational Science Problems. In *Proc. of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin*. Department of Computer Science, University of Tennessee, Knoxville, 1997.

[6] A. Cline. Scalar- and Planar-Valued Curve Fitting Using Splines Under Tension. *Communications of the ACM*, 17:218–220, 1974.

[7] D. Young, D. Kincaid, J. Respess, and R. Grimes. Itpack2c: a FORTRAN package for solving large sparse linear systems by adaptive accelerated iterative methods. Technical report, University of Texas at Austin, Boeing Computer Services Company, 1996.

[8] J. Moré, B. Garbow, and K. Hillstrom. Minpack : Documentation file accessible at: "http://www.netlib.org/minpack/readme".

[9] P. Swarztrauber. FftPack : Documentation file accessible at: "ftp://ftp.ucar.edu/ftp/dsl/lib/fftpack/readme".

[10] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Second Edition*. SIAM, Philadelphia, PA, 1995.

[11] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 5:308–325, 1979.

[12] J. Dongarra, J. Du Croz, S Hammarling, and R. Hanson. An Extended Set of Fortran Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–32, 1988.

[13] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[14] R.W. Freund and N.M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[15] H Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proc. of Supercomputing'96, Pittsburgh*. Department of Computer Science, University of Tennessee, Knoxville, 1996. to appear in *The International Journal of Supercomputer Applications and High Performance Computing*.

[16] H. Casanova, J. Dongarra, and K. Seymour. Client User's Guide to Netsolve. Technical Report CS-96-343, Department of Computer Science, University of Tennessee, 1996.

[17] Inc. Sun Microsystems. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., jun 1987.

[18] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[19] I. Graham. *Object Oriented Methods*. Addison-Wesley, Berkeley, CA, 2d edition, 1994.

[20] S. I. Feldman, D. M. Gay, M. W. Maimone, and N. L. Schrye r. A Fortran-to-C converter. Computing science technical report no. 149, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[21] L. E. Malvern. *Introduction to the Mechanics of a Continuous Medium*. Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

[22] J. C. S. Long, J. S. Remer, C. R. Wilson, and P. A. Witherspoon. Porous Media Equivalents for Networks of Discontinuous fractures. *WRR*, 18:645–658, 1982.

[23] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, Reading, MA, 1988.

[24] J. R. Levine. *lex & yacc.* O'Reilly and associates, Cambridge, MA, 2d edition, 1992.

[25] J. Meyer and T. Downing. *Java Virtual Machine.* O'Reilly & Associates, Sebastopol, CA, 1997.

[26] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification.* Addison-Wesley, Berkeley, CA, 1997.