Using Java and Linda for Parallel Processing in Bioinformatics for Simplicity, Power and Portability

George Wells   Tim Akhurst

Abstract—**This paper discusses the use of Java and the Linda coordination language (specifically, IBM's TSpaces implementation) for the construction of parallel processing applications in the field of bioinformatics. Much existing work in this field makes use of scripting languages such as Perl. Java provides a more powerful programming and deployment environment with strong support for processing genetic data. Furthermore, the Linda approach greatly simplifies the parallelization of bioinformatics applications. Results of a DNA string searching application show very favorable performance benefits arising from the use of Java and TSpaces on a network of commodity workstations. Near-linear speedup is observed for configurations of up to 15 processors, and execution times are reduced by up to 96%.**

I. Introduction

Much of the software development in the field of bioinformatics is done using scripting languages such as Perl. The project described in this paper is an in-depth investigation of the suitability of mainstream programming techniques for the development of parallel bioinformatics applications. The programming language used is Java[1], together with a commercially developed parallel programming framework based on the Linda[2] coordination language, namely TSpaces from IBM[1].

A. Java and Linda

Java is a widely-used, general-purpose programming language, developed by Sun Microsystems in the mid-1990's. It has rapidly become the programming language of choice in many different areas, and has been the subject of extensive language and library development. In particular, it has strong support for string-handling and pattern-matching, provided by specialized libraries. Recent developments have led to increasing interest in Java as a language for scientific high-performance computing (HPC)[2], [3], [4].

One of the major strengths of Java is its portability, allowing Java programs to be executed on different hardware and operating system platforms without the need for recompilation. All that is required is a Java Virtual Machine (JVM), or interpreter, for the specific combination of hardware and operating system being used. While Linux was used for this project, there is no reason why a network of Windows PCs could not be used, or even a heterogeneous collection of various types of PCs and scientific workstations running a wide variety of different operating systems.

TSpaces is based on the Linda coordination language developed by David Gelernter at Yale in the mid-1980's[5]. After an initial period of intense interest in this model, it lost popularity until the development of a number of Java implementations (both commercial and research projects) in the late 1990's. In particular, Sun Microsystems developed the JavaSpaces specification[6] as a component of the Jini system[7]. Several other companies have subsequently developed full commercial implementations of the JavaSpaces specification[8], [9]. Independently, IBM developed TSpaces as a commercial product, based on the Linda programming model[1], [10]. In addition to these commercial developments, a large number of research projects have focused on Java implementations of Linda. A brief overview of the Linda programming model is given in Section II below.

B. Bioinformatics

Bioinformatics is a relatively new field arising from the application of computers to various biological problems, particularly with regard to research on genes and proteins. A recent paper by Cohen provides a concise overview of the current

[1]Java is a registered trademark of Sun Microsystems Inc.

[2]Linda is a registered trademark of Scientific Computing Associates.

state of bioinformatics, and its relationship to both biology and computer science as the foundational disciplines[11].

The following definition for the term *bioinformatics* has been proposed:

> Bioinformatics is conceptualising biology in terms of molecules and applying "informatics techniques" to understand and organise the information associated with these molecules, on a large scale. In short, bioinformatics is a management information system for molecular biology[12].

While not wishing to delve too far into the realms of biology and genetics, a brief overview of some basic molecular biology may be in order. The basic starting point is DNA, which is composed of four deoxyribonucleotides, or *bases*: adenine, thymine, cytosine, and guanine. These compounds are usually referred to by the abbreviations of their initial letters: A, T, C and G, respectively. A triplet of bases is called a *codon*, and encodes the information required to create an *amino acid*. There are twenty amino acids, which may be also be represented by single letter abbreviations. These amino acids combine to form sequences, giving rise to *proteins*.

From this brief description, it should be clear that DNA and protein sequences can easily be represented as strings of letters describing the component parts. At this level, string comparisons and string searching algorithms are essential in bioinformatics. There is, of course, *much* more than this to bioinformatics, which includes many other categories of problems, such as deriving three-dimensional protein structures, etc. Further information may be found in the many references on this subject, such as [13].

While Java has been used in some bioinformatics applications, and is supported by the development of the BioJava libraries[14], [15], to date much of the software development in this area has used scripting languages such as Perl[16]. While such scripting languages are relatively simple to use and have strong support for string-matching operations and for combining other applications, they suffer from limitations such as poor performance due to their interpreted nature and a lack of support for parallel and distributed programming. This has led to growing interest in Java for biological applications. Notable among these is the work of Sheil[17], who developed a system to predict the secondary structure of selected proteins using artificial neural networks in parallel. This system utilized Java and the JavaSpaces implementation of Linda, and hence has much in common with our investigation.

The focus of the research conducted in this project was the identification and location of specific patterns in DNA that correspond to proteins of interest. This involves the application of pattern-matching techniques to strings representing both the DNA sequences and the *motifs* corresponding to the protein sequences of interest. Given the large volumes of data involved (the complete human genome comprises about 4GB of data when represented as a text string), parallel programming becomes a desirable, even necessary, technique for improving the performance of such bioinformatics applications.

The remainder of this paper provides an overview of the Linda programming model, a description of our application and some performance results that were obtained. We conclude with some observations about the use of Java and TSpaces for bioinformatics.

II. The Linda Programming Model

The Linda programming model has a highly desirable simplicity for writing parallel or distributed applications. As a *coordination language* it is responsible solely for the coordination and communication requirements of an application, relying on a *host language* (i.e. Java in this study) for expressing the computational requirements of the application.

The Linda model comprises a conceptually shared memory store (called *tuple space*) in which data is stored as records with typed fields (called *tuples*). The tuple space is accessed using five simple operations[3]:

`out` Outputs a tuple from a process into tuple space

[3]A sixth operation, `eval`, used to create an *active tuple*, was proposed in the original Linda model as a process creation mechanism, but can easily be synthesized from the other operations, with some support from the compiler and runtime system, and is not present in any of the commercial Java implementations of the Linda model.

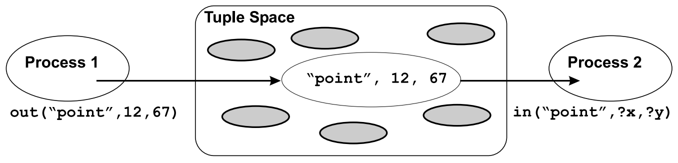

Fig. 1. A Simple One-to-One Communication Pattern

`in`	Removes a tuple from the tuple space and returns it to a process, blocking if a suitable tuple cannot be found
`rd`	Returns a copy of a tuple from the tuple space to a process, blocking if a suitable tuple cannot be found
`inp`	Non-blocking form of `in` — returns an indication of failure, rather than blocking if no suitable tuple can be found
`rdp`	Non-blocking form of `rd`

Note that the names used for these operations here are the original names used in Yale's Linda research. Both TSpaces and JavaSpaces use different names for the operations.

Input operations specify the tuple to be retrieved from the tuple space using a form of *associative addressing* in which some of the fields in the tuple (called an *antituple*, or *template*, in this context) have their values defined. These are used to find a tuple with matching values for those fields. The remainder of the fields in the antituple are variables which are bound to the values in the retrieved tuple by the input operation (these fields are sometimes referred to as *wildcards*). In this way, information is transferred between two (or more) processes.

A simple one-to-one message communication between two processes can be expressed using a combination of `out` and `in` as shown in Fig. 1. In this case `("point", 12, 67)` is the tuple being deposited in the tuple space by Process 1. The antituple, `("point", ?x, ?y)`, consists of one defined field (i.e. `"point"`), which will be used to locate a matching tuple, and two wildcard fields, denoted by a leading `?`. The variables `x` and `y` will be bound to the values 12 and 67 respectively, when the input operation succeeds, as shown in the diagram. If more than one tuple in the tuple space is a match for an antituple, then any one of the matching tuples may be returned by the input operations.

Other forms of communication (such as one-to-many broadcast operations, many-to-one aggregation operations, etc.) and synchronization (e.g. semaphores, barrier synchronization operations, etc.) are easily synthesized from the five basic operations of the Linda model. Further details of the Linda programming model can be found in [18].

III. The Parallel Motif Searching Application

In order to investigate the benefits of using Java and Linda for bioinformatics a large, parallel application was developed to search DNA sequences for subsequences corresponding to specific proteins (or *motifs*). This required the reverse-translation of protein sequences to obtain regular expressions describing the equivalent DNA sequences. Once these had been obtained, the regular expressions could be used with the Java `java.util.regex` package (part of the standard Java libraries since version 1.4 was released) in order to locate the motifs within DNA sequences.

The reverse-translation process involves several steps and is described in more detail below, followed by a description of the design of our application.

A. Obtaining Regular Expressions from Protein Sequences

The first step was to obtain suitable protein sequences. The consensus patterns for a number of protein motifs were downloaded from the PROSITE database[19]. The Sequence Manipulation Suite, developed by the Center for Computational Genomics at Pennsylvania State University[20], was then used for the actual reverse translation, producing a DNA sequence. This was then expressed as a regular expression, suitable for use in a searching algorithm.

For example, the consensus pattern for the protein sequence for the "Homeobox" engrailed-type protein is `L-M-A-[EQ]-G-L-Y-N` (the letters here represent various amino acids). This is then stripped of all characters other than the amino acid specifiers, yielding `LMAEQGLYN`, and reverse-translated using the Sequence Manipulation Suite. This process results in considerable redundancy as a particular amino acid may be produced by a number of codons. For example, the amino acid represented by `L` may have either a `T` or `C` in the first position of the codon, a T in the second position and any one of the four DNA bases in the third position — this particular combination can be represented by the regular expression `[TC]T[GATC]`. The final result is the regular expression corresponding to the "Homeobox" engrailed-type protein:

```
[TC]T[GATC]ATGGC[GATC][GC]A[GA]GG
[GATC][TC]T[GATC]TA[TC]AA[TC]
```

Of particular interest in this expression is the subsequence `[GC]A[GA]`. This is the translation of the `[EQ]` protein sequence, formed from the combination of the regular expressions for the `E` and `Q` amino acids (`E` = `GA[GA]` and `Q` = `CA[GA]`).

B. The Design of the Parallel Motif Searching Application

The motif searching application was developed using the classic replicated-worker pattern[6]. In this case, the master/controller process deposits tuples into the tuple space that specify the work to be done by the remainder of the processes (i.e. the "workers"). Each tuple contains the name of the file containing the DNA sequence to be scanned, a unique task number and the total number of tasks generated (usually, but not necessarily, equal to the number of worker processes). The task number, and total number of tasks are used by the worker processes to determine which segment of the DNA sequence is to be searched. The calculation of the start and end points of the segment also has to include a degree of overlap between adjoining sections to allow for motifs that may span two segments.

After retrieving a tuple with the task parameters, a worker process reads in the calculated segment of the specified DNA sequence from the given file (using the Java random access file facilities). The regular expressions for the motifs of interest are then read in, and the `java.util.regex` library used to perform the searching of the DNA segment. The workers take the results found (start- and end-points of identified motifs, and the actual DNA sequences for the motifs) and return these to the master process as tuples.

The master process retrieves the result tuples, and is then responsible for collating them, and detecting and removing any duplicates found (due to the overlapping of adjoining segments, described above). Finally, the results are summarized and reported (together with processing time taken) to the user of the program.

IV. Experimental Performance Results and Discussion

The performance of the Parallel Motif Searching application was measured for various different sizes of DNA sequences (ranging from 60MB to 1GB), and for various numbers of worker processes. With the exception of the 60MB DNA sequence, it was found that the volume of data was too large to be handled by a single process and so even when only a single worker process was used for the larger sequence, the tasks had to be divided into a number of subtasks. This fact underscores the need for the parallelization of this application.

A. Experimental Environment

The network of workstations that was used for the experimental work consisted of a number of commodity PCs, such as might typically be found in any university or research center. These machines were all identical, equipped with Intel Pentium 4 processors, running at 2.4Ghz, and with 512MB of RAM. A fast (100MBps), switched Ethernet network was used for the communication. Each machine ran the Red Hat Linux operating system, version 3.1.10, and the Java Development Kit used was version 1.4.2_03. The DNA sequences were stored on a central file server and accessed using a shared network drive. Each experiment was repeated a number of times and the average of the results calculated, to minimize the effect of any variations due to other network traffic, or other factors.

B. Results

The results found for each of the DNA sequences followed a similar pattern. Only the largest sequence (i.e. the 1GB file, comprised of the first five human chromosomes) is reported on in detail here (full details are available in [21]). Fig. 2 shows the speedup[4] measured for various numbers of processors. As can

[4]The speedup is calculated as the ratio of the single-worker time to that of the n-worker time. A related factor is *efficiency*: the ratio between the speedup and number of processors, usually expressed as a percentage.

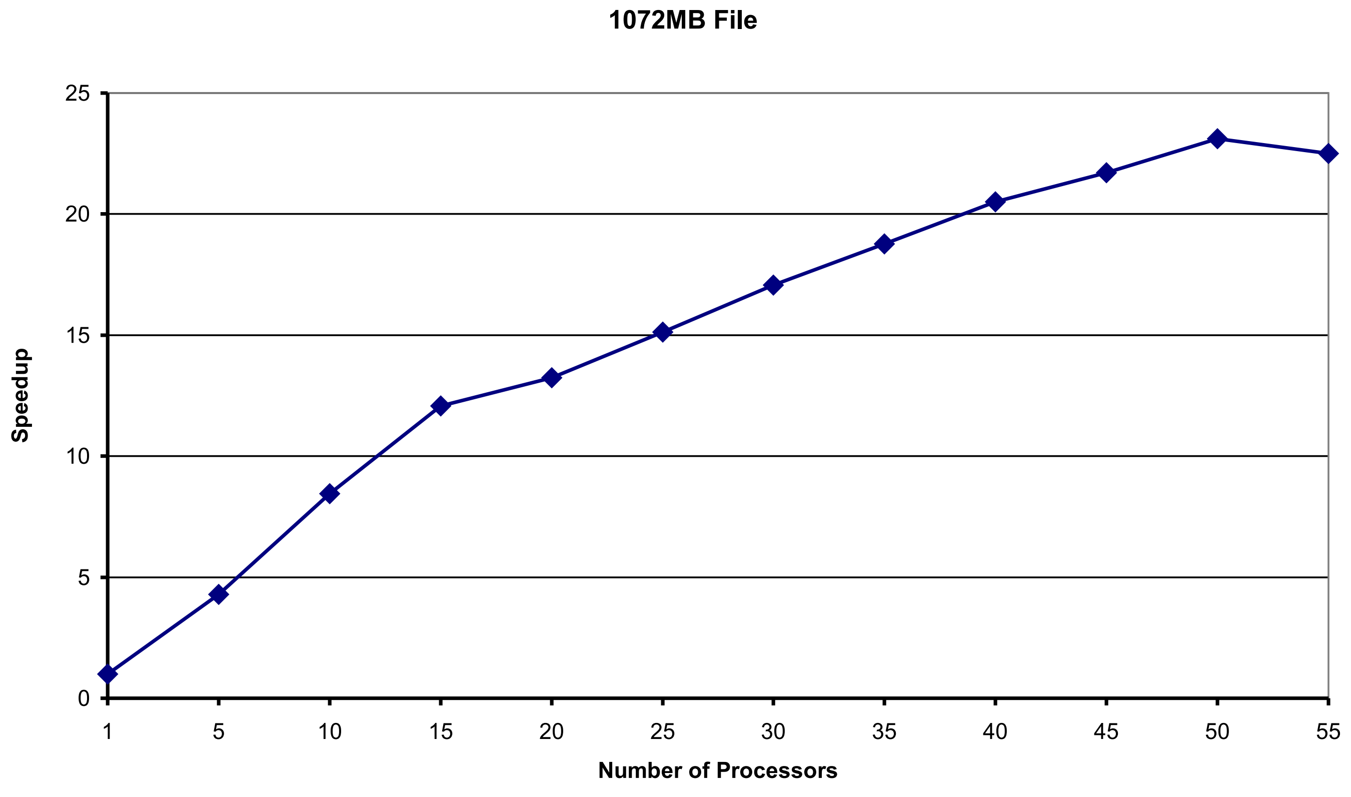

Fig. 2. Speedup for the 1GB DNA Sequence

be seen quite clearly, there is near-linear speedup (i.e. close to 100% efficiency) for configurations up to about 15 processors. Above this level the efficiency starts to decrease, but increasing speedup is still experienced until the number of processors exceeds 50. Using 50 processors, the efficiency is 46%.

In absolute terms, the time taken by a single worker process was 4 hours, 48 minutes (this required using 20 separate segments to accommodate the volume of data). This was reduced to a minimum of 12 minutes and 27 seconds, when using 50 processors — a highly significant reduction in overall processing time. Table I gives the details of the time taken for various numbers of processors.

TABLE I
AVERAGE EXECUTION TIMES FOR THE 1GB GENOME USING A RANGE OF PROCESSORS

Number of	**Average Time**			
Processors	**h**	**m**	**s**	**ms**
1	4	48	06	606
5	1	07	13	811
10		34	05	541
15		23	50	925
20		21	46	214
25		19	02	972
30		16	52	907
35		15	20	859
40		14	03	161
45		13	15	599
50		12	27	503
55		12	48	264

C. Discussion

These results compare well with similar studies. For example, Kleinjung *et al* investigated the use of parallel computing in bioinformatics for performing multiple sequence alignments[22]. Their system was developed in C, using MPI (the Message Passing Interface)[23] for parallelization. They reported that their parallel program performed up to ten times faster on 25 processors compared to the single processor version (giving an efficiency rating of only 40%). The equivalent speedup measured for our application on 25 processors was 15 (i.e. an efficiency of 60%).

Sheil, using Java and JavaSpaces for predicting protein structures using neural networks, reports a speedup of 7.6 times on eight processors[5][17], giving an efficiency measure of 95%. Again, this compares well with our results: using ten processors (the closest comparable datapoint), the speedup that we measured was 8.4 times for the 1GB sequence (which still requires each worker to process a number of separate segments, due to the space limitations discussed above); for the 250MB sequence, we measured a speedup of 9.1, using ten processors.

With regard to the various implementations of the Linda model in Java, Shiel makes the observation that "Jini and JavaSpaces are not easy to use in a rapid development environment — and there is no reason why they should not be"[17]. In another, earlier study of Linda systems in Java, we have drawn similar conclusions[27]. In this regard, TSpaces is a much simpler system to use than JavaSpaces.

V. Conclusions

This study has demonstrated that Java provides a powerful programming and runtime environment, with good support for bioinformatics applications. This is particularly true with regard to string-matching operations using regular expressions, where the `java.util.regex` package is very useful. Furthermore, the Linda programming model, embodied in TSpaces, greatly simplifies the parallelization of bioinformatics applications. At the outset of this project, the author responsible for the development of the software had very little prior programming experience, and none in the field of distributed/parallel programming. That this project was successfully completed in less than a year indicates the ease with which these tools may be deployed.

While scripting languages like Perl have good support for regular-expression matching and basic string-processing operations, and are good at linking together other applications, such environments do not usually have as extensive support for aspects such as parallel and distributed execution of algorithms. A quick survey of the World-Wide Web revealed only a version of PVM[24] for Perl[25], dated 1996, and a form of distributed shared memory (loosely related to the Linda model)[26], which underscores this point.

[5]This was the largest network used in his work.

Results of the DNA motif searching application show that very good performance benefits may be derived from the use of parallel/distributed programming techniques, using a network of commodity workstations, such as those commonly found in most universities or research centers. As indicated in the discussion of results in the previous section, the performance results found in this study are comparable with, or better than results reported in similar studies.

While this study used a dedicated network of workstations for the parallel execution of the application, this is unlikely to be available at all times in a typical research situation. In this regard, Gelernter's research group at Yale did some work on a variation of Linda that provided *adaptive parallelism*[28], [29]. This is a technique that makes use of idle time on workstations in a research group or workgroup. The technique is adaptive, in that, if the user of a PC or workstation starts to make extensive use of the computer, then the parallel framework automatically backs off, ensuring that the background use of the machine for a parallel/distributed processing application does not become intrusive. Similar work has been done in projects such as the well-known SETI screensaver[30]. Techniques such as these would be very useful in extending the practical viability of our work.

As the field of bioinformatics continues to develop and mature, it is our belief that increasing use will have to be made of development tools and languages, such as Java, that have wide adoption in the computer industry, and consequent strong support from commercial and open-source developers and researchers. The wide availability of packages such as TSpaces and JavaSpaces for parallel and distributed programming in these language environments will further encourage developments in this regard. Scripting languages such as Perl will no doubt continue to have a vital role to play, but increasingly this will be confined to the area of combining other applications and interfacing between them.

Acknowledgments

This work was supported by the Distributed Multimedia Centre of Excellence in the Department of Computer Science at Rhodes University, South Africa, with funding from Telkom SA, Business Connexion, Comverse, Verso Technologies and THRIP. Financial support was also received from the National Research Foundation (NRF) of South Africa, from the National Bioinformatics Network (NBN) of South Africa, and from the Department of Biochemistry, Microbiology and Biotechnology at Rhodes University.

References

[1] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford, "T Spaces," *IBM Systems Journal*, vol. 37, no. 3, pp. 454–474, 1998.

[2] C. Pancake and C. Lengauer, "High-performance Java," *Comm. ACM*, vol. 44, no. 10, pp. 98–101, Oct. 2001.

[3] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman, "Benchmarking Java against C and Fortran for scientific applications," in *JGI '01: Proc. 2001 Joint ACM-ISCOPE Conference on Java Grande*. ACM Press, 2001, pp. 97–105.

[4] J. Villacis, "A note on the use of Java in scientific computing," *SIGAPP Appl. Comput. Rev.*, vol. 7, no. 1, pp. 14–17, 1999.

[5] D. Gelernter, "Generative communication in Linda," *ACM Trans. Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[6] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley, 1999.

[7] Sun Microsystems. Jini connection technology. [Online]. Available: `http://www.sun.com/jini`

[8] GigaSpaces Technologies Ltd. (2001) GigaSpaces. [Online]. Available: `http://www.gigaspaces.com/index.htm`

[9] Intamission Ltd. (2003) AutevoSpaces: Product overview. [Online]. Available: `http://-www.intamission.com/downloads/datasheets/-AutevoSpaces-Overview.pdf`

[10] IBM. TSpaces. [Online]. Available: `http://-www.almaden.ibm.com/cs/TSpaces/index.html`

[11] J. Cohen, "Computer science and bioinformatics," *Comm. ACM*, vol. 48, no. 3, pp. 72–78, Mar. 2005.

[12] N. Luscombe, D. Greenbaum, and M. Gerstein, "What is bioinformatics? A proposed definition and overview of the field," *Method Inform Med*, vol. 40, pp. 346–358, 2001. [Online]. Available: `http://-papers.gersteinlab.org/papers/whatis-mim/`

[13] C. Gibas and P. Jambeck, *Developing Bioinformatics Computer Skills*. O'Reilly, Apr. 2001.

[14] BioJava. [Online]. Available: `http://www.biojava.org/`

[15] S. Meloan. (2004, June) BioJava – Java technology powers toolkit for deciphering genomic codes. [Online]. Available: `http://java.sun.com/developer/-technicalArticles/javaopensource/biojava/`

[16] Open Bioinformatics Foundation. bioperl.org. [Online]. Available: `http://www.bioperl.org/`

[17] H. Sheil, "Distributed scientific computing in Java: observations and recommendations," in *PPPJ '03: Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*. Computer Science Press, Inc., 2003, pp. 219–222.

[18] N. Carriero and D. Gelernter, *How to Write Parallel Programs: A First Course*. The MIT Press, 1990.

[19] N. Hulo, C. J. A. Sigrist, V. Le Saux, P. S. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. De Castro, P. Bucher, and A. Bairoch, "Recent improvements to the PROSITE database," *Nucleic Acids Research*, vol. 32, pp. D134–D137, 2004, database issue.

[20] P. Stothard. (2000) The sequence manipulation suite. Center for Computational Genomics, Pennsylvania State Unversity. [Online]. Available: `http://www.cbio.psu.edu/sms/`

[21] T. Akhurst, "The role of parallel computing in bioinformatics," Master's thesis, Rhodes University, 2004.

[22] J. Kleinjung, N. Douglas, and J. Heringa, "Parallelized multiple alignment," *Bioinformatics*, vol. 18, no. 9, pp. 1270–1271, 2002. [Online]. Available: `http://bioinformatics.oupjournals.org/-cgi/content/abstract/18/9/1270`

[23] R. Hempel, "The MPI standard for message passing," in *High-Performance Computing and Networking, International Conference and Exhibition, Proceedings, Volume II: Networking and Tools*, ser. Lecture Notes in Computer Science, W. Gentzsch and U. Harms, Eds., vol. 797. Springer-Verlag, 1994, pp. 247–252.

[24] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*, ser. Scientific and Engineering Computation, J. Kowalik, Ed. MIT Press, 1994.

[25] E. Walker. (1996) Pvm-Perl extension for the Parallel Virtual Machine (PVM) message passing system. [Online]. Available: `http://www.csm.ornl.gov/pvm/perl-pvm.html`

[26] N. Matloff, "PerlDSM: A distributed shared memory system for Perl," in *2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2002)*, 2002. [Online]. Available: `http://-heather.cs.ucdavis.edu/˜ matloff/PerlDSM/-Papers/PDPTA/Paper.pdf`

[27] G. Wells, A. Chalmers, and P. Clayton, "A comparison of Linda implementations in Java," in *Communicating Process Architectures 2000*, ser. Concurrent Systems Engineering Series, P. Welch and A. Bakkers, Eds. IOS Press, Sept. 2000, vol. 58, pp. 63–75.

[28] N. Carriero, D. Gelernter, D. Kaminsky, and J. Westbrook, "Adaptive parallelism with Piranha," Yale University, Tech. Rep. 954, 1993.

[29] D. Kaminsky, "Adaptive parallelism with Piranha," Ph.D. dissertation, Yale University, May 1994.

[30] U. of California. (2005) SETI@home. [Online]. Available: `http://setiweb.ssl.berkeley.edu/`