Implementing an Agent Based Auction Model on a Cluster of Inexpensive Heterogenous Workstations

Timothy M. Lynar, Ric D. Herbert and William J. Chivers

Faculty of Science and Information Technology, The University of Newcastle, Ourimbah, NSW, 2258, Australia.
E-Mail: timothy.lynar@newcastle.edu.au

Keywords: *Agent based model, Cluster, Parallel computing*

EXTENDED ABSTRACT

Agent based models are computer based simulations that model a situation or phenomena from the bottom up using 'agents'. Agent based models are often computationally intensive as they tend to simulate micro behaviours in order to examine the emerging system-level phenomena. Historically the size and scope of agent based models has been limited by a lack of accessible high performance computing equipment. For example, it would have been considered impossible to simulate the behaviour of many independent heterogenous economic agents in the development of early macroeconomic models, so that system level variables like GDP were modelled.

Clustering techniques allow for the creation of high performance computing environments at a relatively low cost and make it possible to now consider modelling the lower-level agents of agent based models. Clusters comprise of independent and self supporting computing systems that are networked together to provide a means of interaction which can be utilised in the completion of common tasks. Clusters work by distributing the processing load of an application amongst its stand-alone computing elements. The stand-alone computing elements of a cluster can be anything from powerful servers to humble personal computers or even a mix of both. For this reason it is feasible to develop a cluster for a fraction of the cost of a supercomputer. This paper looks at the performance of a cluster comprised of heterogeneous recycled personal computers.

The authors evaluate the programming and implementation of an agent-based model on a cluster of inexpensive heterogeneous workstations. The workstations are recycled PCs and this paper concentrates on a Java based solution: JavaParty. This is inexpensive high performance computing as the PCs were previously scrap and all software to run the model comes from the Free and Open Source Software movement.

An existing agent based model of resource allocation through an auction between sellers and buyers is used as the basis for all models in this paper. The model used simulates a simple continuous double auction where buyers and sellers post their bids on a billboard in each round and a trade occurs whenever a bid is greater than or equal to an ask. The model is considered as a game which is repeated over time.

The authors wrote the software for the agent based models used. In this paper current methodologies and frameworks are reviewed and some of the available tools for parallel programming and clustering are discussed. Some preliminary results are presented on the effectiveness of parallel computing methods, in relation to the implementation of the agent based model on a cluster by comparing the execution times of the model. The execution time of two versions of a model are then compared on a larger cluster to compare the effectiveness of the cluster when executing models of both high and low computational intensity.

Some results show a significant improvement in performance can be gained by using the cluster. The results showed that the computational load of the agent based models is important when clustering. If the computational load is trivial then the cluster will not increase the execution speed of the model, and may even decrease it. However if the agent based model has computationally intense threads it can benefit significantly from being implemented and executed on a cluster. The results with computationally complex models showed that the cluster provided a substantial performance boost.

The results are promising and highlight the potential of using cluster to execute agent based models. Clustering is suited to complex agent based models. High performance computing can be made available to those who are researching agent based or individual based models at a relatively low cost through the use of clusters.

1 INTRODUCTION

Agent based models are computer based simulations. Agent based models are often computationally intensive as they tend to simulate micro behaviours in order to examine the emerging system-level phenomena. Historically the size and scope of agent based models has been limited by a lack of accessible high performance computing equipment. For example, it would have been considered impossible to simulate the behaviour of many independent heterogenous economic agents in the development of early macroeconomic models, so that system level variables like GDP were modelled.

Modern computing power and recently the techniques of parallel programming and clustering techniques allow for the creation of high performance computing environments at a relatively low cost and make it possible to now consider modelling the lower-level agents. Parallel programming allows for computing tasks to be performed in a parallel rather than a sequential fashion. Clusters are groups of stand-alone computing elements that usually reside in the same location.

In this paper the authors evaluate the programming and implementation of an agent-based model on a cluster of inexpensive heterogeneous workstations. The workstations are recycled PCs in a Linux based cluster and this paper concentrates on a Java based solution: JavaParty. This is inexpensive high performance computing as the PCs were scrap and all software to run the model comes from the Free and Open Source Software movement. The authors wrote the software for the agent based model. In this paper current methodologies and frameworks are reviewed and some of the available tools for parallel programming and clustering are discussed. The paper then presents some preliminary results of the effectiveness of parallel computing methods, in relation to the implementation of the agent based model on a cluster by comparing the execution times of the model.

An existing agent based model of resource allocation through an auction between sellers and buyers is used. The difference in performance is demonstrated, before and after the implementation of the model on a cluster.

2 AGENT BASED MODELS AND CLUSTERS

2.1 Agent based models and the need for high performance computing

An agent based model simulates micro behaviours in order to investigate the emerging macro phenomena. Agents are software processes that are implemented on a computer and have autonomy, the ability to interact with other agents and the environment, and are both proactive and reactive. Agents are not controlled by any external coordinating device (Gilbert & Terna 2000, Tesfatsion 2002). Agents can be implemented as software objects with their own attributes and methods.

Agent based models have the potential to be processor intensive applications. For example, Goldstein (2007) reports one application involved in modelling the movement of sixteen thousand chickens in a pen. The processor power required to run this simulation was described as "... a chore that would tax a rack full of conventional servers" (Goldstein 2007, p.37). So agent based models raise the issue of the enormity of computing power required to run them in a timely fashion.

Historically, supercomputers have been employed in processor intensive applications such as computer modelling, however this high performance is costly. Supercomputers have the highest cost of any system and yield the highest performance feasible (Schneck 1990). In this paper cheaper alternatives are considered. Clustering promises to be one way of obtaining the computing power of a supercomputer at a relatively low price (Sterling et al. 1999).

2.2 What is a cluster?

Clusters are groups of stand-alone computing elements that usually reside in the same location. Clusters comprise independent and self supporting computing systems that are networked together to provide a means of interaction which can be utilised in the completion of common tasks (Sterling et al. 1999). Clusters work by distributing the processing load of an application amongst its stand-alone computing elements; these elements then process the application. The stand-alone computing elements of a cluster can be anything from powerful servers to humble personal computers or even a mix of both (Sterling et al. 1999). For this reason it is feasible to develop a cluster for a fraction of the cost of a supercomputer. This paper looks at the performance of a cluster comprised of heterogeneous recycled personal computers.

2.3 How does a cluster work?

For a cluster to operate an application must be coded by a programmer in a way that allows it to be distributed amongst the nodes of the cluster. When the application is executed on the cluster these nodes process their portion of the application. In order for a cluster to show a performance increase over a standard desktop computer it has to execute the parts of the

application simultaneously on the different nodes of the cluster. This is known as distributed parallel computing. There are many methods of implementing distributed parallel computing on a cluster. The next section is a review of some of these methods.

3 CURRENT CLUSTER TECHNOLOGY

3.1 Methods of Clustering with Java

For this paper the model was implemented in Java. For a program to be distributed amongst the nodes of a cluster and its parts executed in parallel, specialist solutions must be implemented. There are many such solutions for the creation of a cluster of heterogeneous workstations. Message Passing Interface (MPI) implementations, Remote Method Invocation (RMI) based implementations and OpenMosix are some of the current technologies in this field.

The Message Passing Interface (MPI) is one such solution. MPI is not an application or library but is a standard that was created during 1993–1994 (Gropp & Lusk 1995). MPI is a standard for how processes should communicate once they are created. It does not specify how processes should be created nor does it include dynamic processor management (Gropp & Lusk 1995). There are many implementations of the MPI standard, including M-JavaMPI, and Pure Java Message Passing Implementation (PJMPI) which are implementations for Java.

Java implementations of MPI fall into two categories: Pure Java implementations, which are reported to be slow, and implementations that have implemented native libraries, and as such could not be considered truly portable (Ma et al. 2002). WeiQin et al. appear to agree that native libraries are unsuitable for a heterogenous environment stating "even the programs written in the same MPI binding language run on different platforms may not be able to communicate because of the difference of internal byte ordering" (WeiQin et al. 2000, p.533). For this reason MPI technology was not used for this cluster.

Parallel programming on Java is not exclusively related to MPI. One other technology is Dataflow Java. Dataflow Java is an extension of Java that provides an alternative way of programming multi-threaded applications. Dataflow Java allows applications to be developed in a data centric way (Lee & Morris 2000). Dataflow Java takes control of most of the thread management in an application and claims to be "implicitly parallel" (Lee & Morris 2000, p.1). It is important to note that dataflow Java remains untested and unreleased and as such is unsuitable to be implemented in this project. However the concept behind Dataflow Java is of interest and may be appropriate as it is developed in the future.

The Java Object-Passing Interface (JOPI) extends Java to provide additional functionality. Java objects are utilised to assist communication among parallel applications. The functionality that is provided is similar to data passing extensions such as MPI, but is claimed to be more powerful (Mohamed et al. 2002, Al-Jaroodi et al. 2002). JOPI provides an interface to the programmer that is similar to MPI (Mohamed et al. 2002) and appears to provide greater performance than MPI. JOPI appears to be an excellent technology but has not been used for this project, as it appears to be no longer available.

Titanium (Yelick et al. 1998) is another option for parallel computing. It uses Java as its base and is based on the SPMD principle, (Single Program, Multiple Data). Titanium is not an extension of Java; rather the Titanium compiler translates Titanium's code into C which then needs to be compiled by a C compiler (Yelick et al. 1998). However Titanium does not support threads and is not as cross-platform as JavaParty.

JavaParty (Moschny & Haumacher 2007) is a cross platform parallel programming extension to Java. JavaParty consists of a pre-processor and a runtime-system utilising a remote method invocation interface to enable objects and threads to be spread across the distributed environment. JavaParty is designed for "... distributed parallel programming in heterogeneous workstation clusters" (Philippsen & Zenger 1997, p.1226). The main advantage of JavaParty to the programmer is that there is "... no need to significantly re-write or re-organize a given Java program" (Philippsen & Zenger 1997, p. 1226). Few changes need to be made to a class to make it parallel. To distribute an object to a remote machine the class modifier `remote` is required and instead of using Java's threads the `RemoteThread` class is provided. Using these two simple modifications it is possible to distribute the computing load of an application across all the nodes in a cluster. Because of JavaParty's ease of programming and promising performance results it was chosen as the technology to run the agent-based model on.

JavaParty comes in two versions with each having a different remote method invocation interface. One version is based on Java's RMI and the other is based on KaRMI, which is a replacement for the standard Java RMI. KaRMI was designed for high performance computing and tests have shown that it saves a median of 45% of runtime (Philippsen et al. 2000). The version of JavaParty used in this project utilises KaRMI.

OpenMosix is yet another alternative for clustering. OpenMosix consists of a kernel patch and some user-space tools. OpenMosix allows for the load-balancing of processes among other connected OpenMosix nodes. The main advantage of OpenMosix is that it is implemented in the kernel and user-space and only requires the programmer to program in threads. This simplifies the task of clustering but also creates new problems. The programmer must create or modify a distribution of unix to include the Openmosix kernel patch and tools. OpenMosix is also limiting in that it can only be used on IA architecture and all nodes must be running Linux (Knox 2007). OpenMosix should provide a quick and easy way to cluster heterogenous workstations, providing those workstations are of the IA-32 architecture (Knox 2007). OpenMosix however currently only has alpha support for the 2.6.x series of Linux kernel (Knox 2007). This means that if you wish to use OpenMosix you must use an older distribution of Linux that has an old 2.4.x series kernel.

After reviewing the available software that will assist in the creation of an agent based model on a cluster of heterogeneous workstations, it was decided that JavaParty is currently the best option for the creation of a simple cluster of heterogeneous nodes. JavaParty is a cross-platform Java-based technology that supports threading and can enable remote execution of code with only minor changes to an application.

4 IMPLEMENTATION

4.1 The Cluster

The cluster used consists of eleven nodes. Each node is a stand-alone personal computer. The nodes in the cluster are connected to each other through standard 100Mb Ethernet network interface cards and are connected through Ethernet cables to a 100Mb switch. All the personal computers in the cluster are of the same hardware makeup, they are all about four years old and contain Pentium 4 processors. These pieces of hardware have been selected because they have been written off and were to be disposed of like many other computers their age.

The cluster used in the preliminary tests with model A consisted of two nodes and did have computers of differing hardware, however the slowest node in this cluster was dropped from the final cluster because of its comparatively poor performance.

The software environment of the cluster is currently homogeneous. The Debian GNU/Linux (Software in the Public Interest Inc 2007) distribution of Linux is installed on all the workstations in the cluster. Sun Java 1.4.2-02 run-time environment (Sun 2007) is installed and JavaParty (KaRMI) 1.9.5 is also present. One advantage of using Java is that it is possible to have nodes of differing hardware and software makeup, so long as those nodes can run the Java run-time environment, they should be able to be included in the cluster.

4.2 Measuring performance

To test the performance of the applications on the cluster, the Linux `time` command is used. The `time` command has the ability to record the time an application takes to execute. The `time` command is accurate to one-tenth of a second (Christias 1994).

The values returned from the Linux `time` command have been compared. The speed of the execution of the application has been tested repeatedly with no other non-system specific applications running. All task schedulers were stopped, the swap partition was disabled and the application run in console mode. This application has been tested on the stand-alone system after the application has been modified, to enable it to be run in parallel. Once the cluster was setup, turned on, and all nodes connected, the application was then tested again using the same method as described above. All the results have been recorded.

All data has been tabulated in JMP 5.1 (Institute 2007) and graphs and statistics produced. An ANOVA has been conducted to analyse variance between tests and Tukeys Honestly Significant Difference (HSD) test is used to show significant difference. The mean and median execution times have also been calculated.

4.3 The Agent Based model

The model used in this paper examines a number of simple strategies for agents in an auction. The model allows the agents to evolve with unsuccessful agents dying and being replaced by agents that inherit the most successful strategy. The model seeks to answer the question as to whether a particular strategy dominates or whether a steady state of a mixture of strategies evolves. The auction model used is a continuous double auction where buyers and sellers post their bids on a billboard in each round and a trade occurs whenever a bid is greater than or equal to an ask. The model is considered as a game which is repeated over time. The details of the model are presented elsewhere in this conference in Herbert & Turton (2007).

4.4 Model A

The agent based model that is used as the preliminary model (model A) is a resource allocation model through an auction between sellers and buyers. For the preliminary results the model has been further simplified. It contains a simple `for` loop to create a number of agents each in a software thread. The agents in this model are ZIT (Zero Intelligence Traders) so that each agents either make a bid of a random amount or an ask of a random amount. There was no clearing of the bids implemented in this model. The aim of this simple model is to demonstrate what effect implementing an agent based model on a cluster may have.

4.5 Model B and Model C

The final model is more computationally intensive than the first model and is executed over the full cluster. In the final model clearing occurs and all results are recorded to a database. The final model comes in two versions of varying computational intensity. The first version of the final model (model B) contains the same computationally light model as the preliminary model, however it contains agents of varying type, evolution and clearing. The second version (model C) is the same as the first but with each agent preforming an additional computationally intense task, to simulate the computational impact of more complex agent based models.

5 RESULTS

5.1 Results - model A

Preliminary analysis was performed on a two node cluster. A two node cluster was used to test whether the current setup was correct and whether a reduction in execution time had occurred as a result of clustering. The two node cluster consists of two personal computers of different hardware. The nodes of the cluster were connected together through the available switch and network cables. Java and JavaParty were installed. Initial tests were then conducted on each individual node of this cluster and then the two node cluster itself.

Node 0 appears to be the fastest node as it ran the model in a mean time of 41.54 seconds and a median time of 41.41 seconds. Node 1 ran the model much slower with a mean time of 61.94 seconds and a median time of 57.90 seconds. When the nodes were clustered together they ran the model with a mean time of 36.59 seconds and a median time of 36.83 seconds.

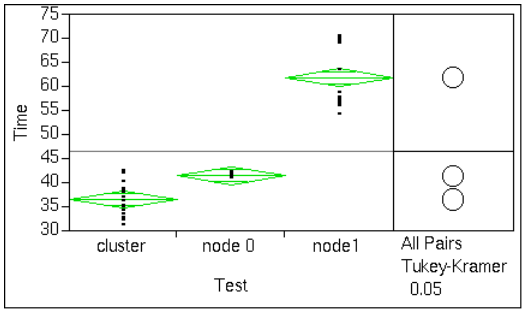

Figure 1. ANOVA of model A

An analysis of variance has been conducted and Tukeys HSD used to show significant difference between tests. Figure 1 shows these results. A one-way Analysis of Variance(ANOVA) was used to compare times of execution over the three possible configurations (node 0, node 1 and cluster). The ANOVA showed a significant difference at the 0.05 level of confidence. Tukeys HSD test was then used to find that there were significant differences between all groups at the 0.05 level of confidence. The results show that the clustering techniques produce a statistically significant overall improvement in performance by producing significantly lower results than that of the fastest individual node (node 0).

The nodes and the cluster were each tested twenty times to determine the execution speeds of the model. These results are displayed in Figure 2. It must be noted the results with a two node cluster were somewhat erratic as were the results of node 1. The operating system, setup and all software was identical on both nodes. The erratic behaviour of node 1 may be attributed to the uneven distribution of threads to the different cores on the dual core processor of node 1. The cluster also showed erratic behaviour and this may also be attributed to the uneven distribution of threads to the different nodes within the cluster.

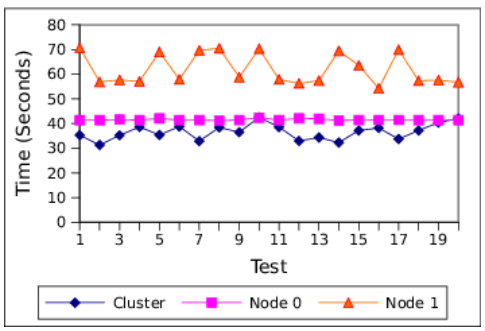

Figure 2. The time taken to execute model A

5.2 Results - model B

Results were derived by running tests on the full cluster. Six tests were conducted with two versions of the final model (model B and C). For each model three tests were conducted. A test was conducted with each model running entirely on a single node (node 0), with the model running on a single node (node 7) with an external database (node 0) and the model run over the entire cluster. Each model was run twenty five times for each test.

The cluster performed poorly when executing model B. When the model was run on a stand-alone workstation the model took a mean time of 40.0 seconds to execute. When the model was executed with a separate computer acting as a database server, the execution time dropped to an mean time of 30.2 seconds. When the model was executed over the entire cluster the execution speed was slightly slower although not significantly slower, with an mean execution time of 31.0 seconds.

Figure 3 shows the results of the execution of model B over the full cluster. Although an ANOVA shows there is significant difference at the 0.05 level of confidence, Tukeys HSD test when used showed that there were significant differences between some groups but not others. The overlapping circles in Figure 3 indicate that the tests for the cluster and node 7 and 0 are not significantly different from each other.

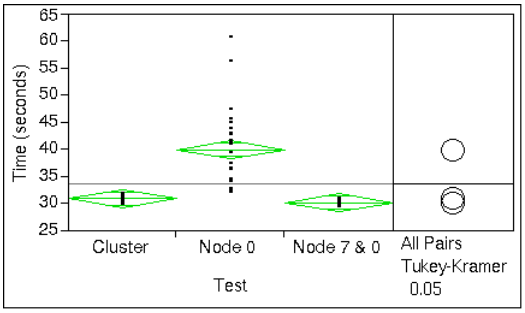

Figure 3. ANOVA and Tukeys HSD of model B

The results from model B show that the cluster provides no significant performance increase if the application executed on the cluster has a light computational function to perform. The ANOVA and Tukeys HSD of model B show that even when the threads of the application have little to do, no significant decrease or increase in performance is realised.

5.3 Results - model C

When model C was executed over a stand-alone workstation it had a mean execution time of 65.7 seconds. When this model was executed on a single computer with an additional computer acting as a database server, the model executed in an mean time of 57.4 seconds. When the model was run on the cluster it executed in an mean time of 33.0 seconds.

An ANOVA was used to compare times of execution over the three possible configurations of the two models (Cluster, node 0, and node 7 and 0). Tukeys HSD test was then used to find wether there were significant differences between all groups at the 0.05 level of confidence.

Figure 4 show the results of model C. The ANOVA shows significant difference at the 0.05 level of confidence and Tukeys HSD test found that all groups were significantly different.

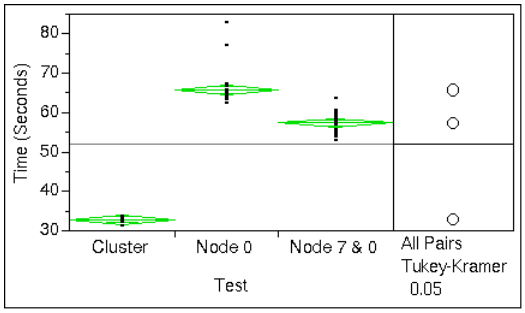

Figure 4. ANOVA and Tukeys HSD of model C

The results from model C show that when the individual threads of the application have a substantial task to perform, the cluster provides a significant speed increase over the fastest stand-alone system.

6 CONCLUSIONS

This paper has presented some preliminary results from running a simple agent based model on a cluster of heterogenous workstations. Initial results indicate that implementing an agent based model on a cluster can lead to significant speed increases in the execution of the model. The implementation of a Java agent based model was easily implemented on a cluster of recycled heterogenous workstations running Linux and JavaParty. All hardware and software used in this cluster were obtained at no cost to the authors of this paper. The workstations were old, written off and destined for disposal. The operating system and supporting software were all free. Although the hardware and software that comprises this cluster is free it should be noted that it can take considerable time to initially acquire the skills and knowledge to setup such a clustering environment. In this case it took one person three weeks to build the initial cluster. However, subsequent setups of clusters using the same technology can be accomplished in a much shorter

time period. The full cluster of eleven nodes took the same person four days to setup.

The final results show that the computational load of the threads in the agent based models is important when clustering. If the computational load of these threads is trivial then the cluster will not increase the execution speed of the model, and may even decrease it. However if the agent based model has computationally intense threads it can benefit significantly from being implemented and executed on a cluster. High performance computing can be made available to those who are researching agent based or individual based models at a relatively low cost through the use of clusters. However these results highlight that clusters are not suitable to all applications. For an application to take full advantage of a cluster of this type the application must have many computationally intense threads that can be executed over the multiple nodes of the cluster.

Future work is planned to extend the test model and expand the number of nodes on the cluster. Further testing is planned to evaluate the full potential of this technology.

7 REFERENCES

Al-Jaroodi, J., Nader, M., Hong, J. & Swanson, D. (2002), An agent-based infrastructure for parallel java on heterogeneous clusters, *in* B. Gropp, R. Pennington, D. Reed, M. Baker, M. Brown & R. Buyya, eds, 'Proceedings 2002 IEEE International Conference on Cluster Computing', Chicago, Illinois, USA, pp. 19–27.

Christias, P. (1994), 'Time(1)', Retrieved May 8, 2007, from `http://unixhelp.ed.ac.uk/CGI/man-cgi?time`.

Gilbert, N. & Terna, P. (2000), 'How to build and use agent-based models in social science', *Mind & Society* **1**(1), 57–72.

Goldstein, H. (2007), 'Cure for the multicore blues', *IEEE Spectrum* **44**, 37–39.

Gropp, W. & Lusk, E. (1995), Dynamic process managment in a mpi setting, *in* 'Proceedings of Seventh IEEE Symposium on Parallel and Distributed Processing', San Antonio, Texas, USA, pp. 530–533.

Herbert, R. D. & Turton, P. (2007), Simple strategies of agents in an evolving auction model, *in* 'Proceedings of MODSIM07'.

Institute, S. (2007), 'Jmp software', retrived 16 June 2007 from `http://www.JMP.com/`.

Knox, B. (2007), 'OpenMosix, an open source linux cluster project', retrieved 4th June 2007 from `http://openmosix.sourceforge.net/`.

Lee, G. & Morris, J. (2000), 'Dataflow java: Implicitly parallel java', *Australasian Computer Architecture conference 2000* **5**, 42–50.

Ma, R. K. K., Wang, C.-L. & Lau, F. C. M. (2002), 'M-javampi: A java-mpi binding with process migration support', *IEEE/ACM International Symposium on Cluster Computing and the Grid* **2**, 255–263.

Mohamed, N., Al-Jaroodi, J., Jiang, H. & Swanson, D. (2002), Jopi: a java object-passing interface, *in* 'JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande', ACM Press, New York, NY, USA, pp. 37–45.

Moschny, T. & Haumacher, B. (2007), 'JavaParty – Java's Companion for Distributed Computing', retrived 16 Jun 2007 from `http://www.ipd.uni-karlsruhe.de/JavaParty`.

Philippsen, M., Haumacher, B. & Nester, C. (2000), 'More efficient serialization and rmi for java', *Concurrency: Practice and Experience* **12**(7), 495–518.

Philippsen, M. & Zenger, M. (1997), 'Javaparty - transparent remote objects in java', *Concurrency: Practice and Experience* **9**(11), 1225–1242.

Schneck, P. B. (1990), 'Supercomputers', *Annual Reviews Computer Science* **4**, 13–36.

Software in the Public Interest Inc (2007), 'Debian – the universal operating system', Retrieved May 28, 2007, from `http://www.debian.org/`.

Sterling, T., Salmon, J., Becker, D. & Savarese, D. F. (1999), *How to Build a Beowulf*, MIT Press, London.

Sun (2007), 'Java technology', Retrieved May 28, 2007, from `http://www.sun.com/java/`.

Tesfatsion, L. (2002), 'Agent-based computational economics:modelling economies as complex adaptive systems', *Information Sciences* **149**(4), 262–268.

WeiQin, T., Hua, Y. & WenSheng, Y. (2000), Pjmpi: Pure java implementation, *in* 'The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region', Vol. 1, IEEE Computer Society, Los Alamitos, CA, USA, pp. 533–535.

Yelick, K., Semenzato, L., Pike, G., Miyamoto, C., Liblit, B., Krishnamurthy, A., Hilfinger, P., Graham, S., Gay, D., Colella, P. & Aiken, A. (1998), 'Titanium: A high-performance Java dialect', *Concurrency* **10**, 825–836.