

# Benchmarking Beowulf Clusters Using High Performance Linpack (HPL) and Cellular Automata

Rafael Saldaña, Allan Espinosa  
High Performance Computing Research Group  
School of Science and Engineering, Ateneo de Manila University  
Loyola Heights, Quezon City, Philippines 1108  
Telephone/ Fax: +63 2 426 6125  
rsaldana@ateneo.edu, allane@acm.org

## ABSTRACT

A Beowulf cluster is a parallel computer using commercial-off-the-shelf switch-based network to communicate among processors. Ateneo de Manila University maintains two Beowulf clusters: AGILA and MEDGRID. This paper deals with benchmarking the AGILA and MEDGRID using High Performance Linpack (HPL) and standard cellular automata (CA) algorithms such as the Game of Life and Fredkin's rule. The MEDGRID cluster demonstrated a peak performance of 40 Gflops while the AGILA cluster peaked at 3.5 Gflops. Results also show that benchmarking Beowulf clusters using CA algorithms is comparable with that of using HPL.

## Keywords

Beowulf cluster, High Performance Linpack (HPL), Cellular Automata, cluster computing, high performance computing, benchmarking, performance evaluation

## 1. INTRODUCTION

A *cluster* refers to independent computers combined into a unified system through software and networking. Clusters are typically used for *High Availability (HA)* for greater reliability or *High Performance Computing (HPC)* to provide greater computational power than a single computer can provide [1].

A Beowulf cluster is a scalable performance cluster based on commodity hardware, on a private system network, with open source software (e.g., Linux) infrastructure. The performance of Beowulf clusters can be improved proportionally with additional nodes or machines. The commodity hardware can be any of a number of mass-market, stand-alone compute nodes, as simple as two computers each running Linux and sharing a file system or as complex as 1024 nodes with a high-speed, low-latency network [1].

In the year 2000, a team of researchers from the mathematics, information systems & computer science, and physics

departments of the School of Science and Engineering of Ateneo de Manila University built a Beowulf cluster to support its Computational Science Initiative [2, 3]. Called AGILA, the cluster was primarily used for teaching parallel computing to students and for faculty and student researches in computational science (including modeling and simulation, coding theory, and numerical weather prediction). From an initial eight compute nodes and one master node, the AGILA cluster has been expanded to 16 compute nodes and one master node. It runs on Fast Ethernet network technology. Today, the master node has the following configuration:

- AMD Athlon 600 MHz Processor
- 128 Mb SDRAM
- 4 Disk Drives totaling 173.4 Gb of Disk space
- 2 Intel Ethernet Express Pro 100+ Fast Ethernet Controller

The slave nodes were also upgraded and now exhibit a heterogeneous structure consisting of

- 11 AMD Athlon 600 MHz Processors
- 2 Dual Intel Pentium III 600 MHz Processors
- 1 Gb of Disk space for the slave node operating system
- 4 128 Mb, 7 64 Mb, 2 512 Mb RAM

In 2003, Ateneo de Manila University acquired another Beowulf cluster as a result of its collaboration with Japan's National Institute of Advanced Industrial Science and Technology (AIST) to pilot a project on the applicability of Grid computing techniques for fMRI analysis [4]. Called MEDGRID, the cluster consists of 8 dual processor compute nodes and one master node, and uses MYRINET network technology [5]. The master node consists of the following specifications:

- Dual Intel Xeon CPU 2.8 0GHz
- 2 Gb RAM
- 3 200 Gb harddisk drive
- Intel 82545EM Gigabit Ethernet Controller
- 3Com 3c905C Fast Ethernet Controller

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

The compute nodes uses a Myrinet-2000 class switch for its compute interconnect. The eight compute nodes has the following configuration:

- Dual Intel Xeon 2.80GHz
- 1 Gb RAM
- 80 Gb harddisk drive
- Intel 82545EM Gigabit Ethernet Controller
- Myricom M3F-PCIXD Myrinet-2000 class controller, Lanai XP Chipset version

## 2. HIGH PERFORMANCE LINPACK (HPL)

*Linpack* is a set of Fortran subroutines that can be used for solving various systems of linear equations [6]. For example given a problem to solve the linear equation.

$$\mathbf{A}x = \mathbf{b}. \quad (1)$$

In Equation 1 the matrix  $A$  will be decomposed into a product of well structured matrices which is easier to manipulate in order to solve the linear equation in 1. The Linpack benchmark was originally a supplement report in the original Linpack User's guide. As the years go by, the content of the performance data increased. The report contains the the performance of solving the general dense matrix 1 in double precision arithmetic. The original benchmark evolved into the *High Performance Linpack* or *HPL* benchmark suite and is used by various institutions to measure the general performance of their supercomputers. The TOP500 project [7] uses the suite to provide a reliable basis for tracking and detecting trends in high performance computing since 1993.

HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. It can thus be regarded as a portable as well as freely available implementation of the High Performance Computing Linpack Benchmark [8].

Below is a description of the HPL's Main Algorithm [9]:

"This software package solves a linear system of order  $n$  :  $Ax = b$  by first computing the LU factorization with row partial pivoting of the  $n$ -by- $n + 1$  coefficient matrix  $[Ab] = [[L, U]y]$ . Since the lower triangular factor  $L$  is applied to  $b$  as the factorization progresses, the solution  $x$  is obtained by solving the upper triangular system  $Ux = y$ . The lower triangular matrix  $L$  is left unpivoted and the array of pivots is not returned.

The data is distributed onto a two-dimensional  $P$ -by- $Q$  grid of processes according to the block-cyclic scheme to ensure 'good' load balance as well as the scalability of the algorithm. The  $n$ -by- $n + 1$  coefficient matrix is first logically partitioned into  $nb$ -by- $nb$  blocks, that are cyclically 'dealt' onto the  $P$ -by- $Q$  process grid. This is done in both dimensions of the matrix (Figure 1).

The right-looking variant has been chosen for the main loop of the LU factorization. This means that at each iteration of the loop a panel of  $nb$  columns is factorized, and the trailing sub-matrix is updated. Note that this computation is thus logically partitioned with the same block size  $nb$  that was used for the data distribution (Figure 2)."

High performance computing facilities measure the of their supercomputers in terms of FLOPS (Floating Point Operations Per Second) [10]. The HPL software package was specifically designed to measure the performance of distributed-memory computers. The first benchmarking study of the

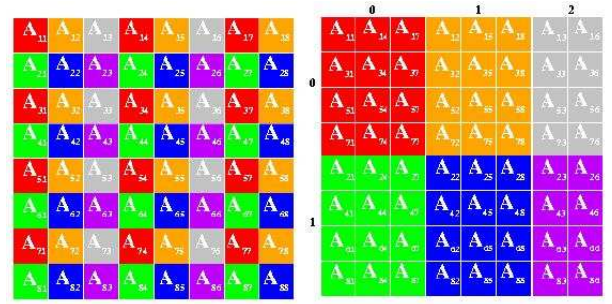


Figure 1:  $P \times Q$  grid of processes for good load balancing

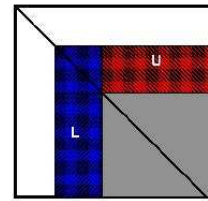


Figure 2: LU Factorization of the matrix

AGILA Beowulf cluster reached a peak performance of 3.16 Gflops [2, 3]. The benchmark provides a testing the timing program to quantify the accuracy of the obtained solution of the matrix and the time it took to compute.

In the current study, the authors used HPL to evaluate the performance of the AGILA and MEDGRID clusters. This paper serves as a second benchmark report of the AGILA cluster. On the other hand, MEDGRID does not yet have any published documentation on its peak performance prior to this study.

BLAS is a collection of basic linear algebra subprograms [11]. To successfully build HPL, a C interface to BLAS was used to be able to use the MPI middleware interface of the clusters. ATLAS (Automatically Tuned Linearly Algebra Software) [12] was used in order to automatically obtain the parameters for efficient linear algebra computations. The software package can be obtained from its SourceForge.net project page [13]. The sourcecode was downloaded as a tarball and built into both MEDGRID and AGILA. According to the INSTALL.txt file of the package, there are two mandatory steps for a successful ATLAS installation: the configuration and building of the source files. The Makefile provides a step by step configuration instructions to properly tune ATLAS for MEDGRID and AGILA. The following lines shows excerpts from a shell session in AGILA for building ATLAS.

```
$HOME>cd /opt
/opt>tar -xjf $HOME/atlas3.6.0.tar.bz2
/opt>cd ATLAS
/opt/ATLAS> make config
./xconfig
ATLAS3.6.0 configure started.
...
Probing to make operating system determination:
Operating system configured as Linux

Probing for architecture:
Architecture is set to ATHLON
```

```
Probing for supported ISA extensions:
  Altivec: NO.
  Altivec: NO.
  SSE2: NO.
  SSE1: NO.
  3DNow2: DETECTED!
```

ATLAS has detected that your machine has 3DNow! capability, and ATLAS can use these instructions to dramatically speed up single precision operations. However, 3DNow! does not use IEEE compliant arithmetic; in particular, it does not handle NaNs or Infinities at all (operations on them are essentially undefined), and it does not handle overflow or underflow correctly. There may be other discrepancies as well. Therefore, only enable 3DNow! if you are very sure that these shortcomings in accuracy do not concern you. In general, getting the answer very fast is no substitute for getting the \*correct\* answer, so just say no if you are at all unsure.

Use 3DNow! for computation? [n]: y

Required cache flush detected as : 1048576 bytes

Looking for compilers (this may take a while):

```
/usr/bin/gcc : v2.95.3
F77 = /usr/bin/g77 -fomit-frame-pointer -O3
CC = /usr/bin/gcc -fomit-frame-pointer -O3 -funroll-all-loops
MCC = /usr/bin/gcc -fomit-frame-pointer -O3
```

ATLAS has default parameters for OS='Linux' and system='ATHLON'.

If you want to just trust these default values, you can use express setup, drastically reducing the amount of questions you are required to answer

```
}
{scriptsize
\begin{verbatim}
use express setup? [y]: y
```

You need to choose a name which represents this architecture (eg. UltraSparc, Dec21164, etc). Do not use a generic name (eg. solaris, linux), which might apply to different hardware. This architecture name will be appended to the name of the created make include file, and appear in all subdirectories, so don't make it longer than you like to type. The name should follow the rules for file names (so don't use punctuation and spaces, for instance).

Enter Architecture name (ARCH) [Linux\_ATHLON3DNow2]: agila  
<arch> set to 'agila'

The ATLAS install process is heavily file-based, and this can cause major reliability problems when interacting with an overloaded or malfunctioning remotely mounted filesystem. ATLAS therefore has a mechanism in place to allow for a delay before a file is declared to not be there, so that slow NFS (i.e., waiting for amd timeout) problems can be overcome, or for handling slightly differing clocks between server/client. This problem is magnified if doing cross-compilation. In the question below, we ask how much of a delay, in seconds, ATLAS should tolerate between file creation and appearance. If you are installing on a local filesystem (eg. /tmp) or a smooth-running NFS system, answer 0; for a moderately loaded NFS server, you may want a value in the 10 second range, and for cross-compiling systems or NFS servers experiencing errors, you may want to go as high as a couple of minutes (120).

Enter File creation delay in seconds [0]: 0

I'm going to ask you for information about your Fortran 77 compiler. ATLAS does not need Fortran77 to build, so if you don't have a Fortran compiler, the install can still be completed successfully. However, ATLAS built without a Fortran compiler will not be callable from Fortran (i.e., the user should use the C interface), and we will not be able to do full testing, since some of the tester code is written in Fortran77.

```
F77 & FLAGS: /usr/bin/g77 -fomit-frame-pointer -O3
FLINKER & FLAGS: $(F77) $(F77FLAGS)
```

```
CC & FLAGS: /usr/bin/gcc -fomit-frame-pointer -O3 -funroll-all-loops
MCC & FLAGS: /usr/bin/gcc -fomit-frame-pointer -O3
CLINKER & FLAGS: $(CC) $(CCFLAGS)
```

Finding F77 to C calling conventions (this may take a while):

```
Calculated F77/C interoperation conventions:
  Suffix F77 names with underscores with __
  F77 INTEGER -> C int
  F77 strings handled via standard sun style
```

The ATLAS team has provided a default install for your architecture. If you want, these default values can be used, and ATLAS can skip most of the search for your machine. This will greatly decrease the amount of time required for the install, allow you to take advantage of any special features found by the ATLAS team, and provide protection against install miscues caused by unreliable timing results, assuming you really have the machine ATLAS thinks you have. If your machine is non-standard in some way, or you just want to see the ATLAS search for yourself, you should answer no to the following question. Otherwise, it is highly recommended to accept the default of yes.

Use supplied default values for install? [y]: y

Unpacking Architectural defaults . . . done.

Creating make include file Make.agila  
Make.agila successfully created.

Creating ATLrun.sh

```
Creating subdirectories:
  Checking for already existing subdirectories . . . . . no
Subdirectories successfully created.
```

Storing L1 cache size of 64KB.

Moving config logfiles ConfSummary.log and ConfDump.log to bin/agila/INSTALL\_LOG/

Configuration completed successfully. You may want to examine the make include file (Make.agila) for accuracy before starting the install with the command:  
make install arch=agila

```
rm -f ./xconfig
```

```
/opt/ATLAS>make install arch=agila
```

After installing the dependencies on the system, HPL was configured based on the `Make.PII_Linux_CBLAS_gm` Makefile from the `hpl/setup` directory. The lines pointing to the MPI libraries were changed to indicate that MEDGRID uses the SCORE Cluster System Software [14] to utilize the Myrinet interconnect hardware. It has its own version of the mpich [15] MPI-1 implementation designed for the system. For AGILA, the pre-existing CBLAS interface that was used in its initial deployment was used.

### 3. CELLULAR AUTOMATA

Cellular automata (CA) are dynamic systems of discrete lattice sites whose behavior are specified in terms of local rules or relations. First introduced by John von Neumann and Stanislaw Ulam in the late 1940s, cellular automata have been used to model successfully many complex phenomena. Cellular automata are characterized by the following:

1. Space is represented by a uniform lattice.
2. Each cell contains some data and is connected to other cells in a particular pattern.
3. time advances in discrete steps.
4. The state of each cell is governed by a set of rules that are applied at each time step.

The general CA rule is that the state of each cell in the next timestep depends on the state of the cell and some other cells in the lattice. The collection of all these cells is called the *neighborhood* of the cell. Some examples of CA neighborhoods are the Von Neumann neighborhood (wherein each cell is surrounded by four nearest neighbors North, East, South, and West) and the Moore neighborhood (wherein each cell is surrounded by eight nearest neighbors North, Northeast, East, Southeast, South, Southwest, West, and Northwest).

Invented by John Conway in the 1960's, he Game of Life [16] is a simple cellular automaton based on the following rules on a Moore neighborhood.

1. Any live cell with fewer than two neighbours dies, as if by loneliness.
2. Any live cell with more than three neighbours dies, as if by overcrowding.
3. Any live cell with two or three neighbours lives, unchanged, to the next generation.
4. Any dead cell with exactly three neighbours comes to life.

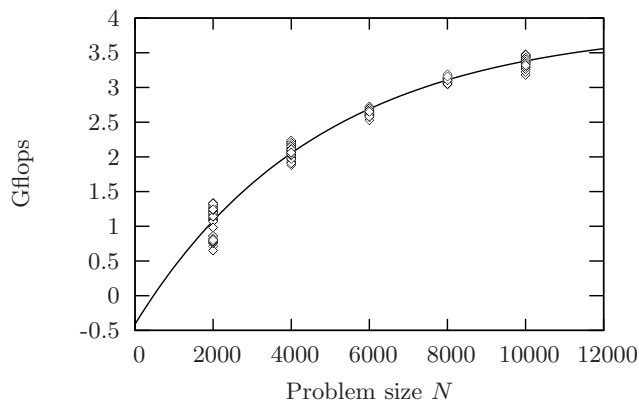


Figure 3: HPL benchmark results to obtain AGILA's peak using 16 processors.

A variant of the Game of Life, Fredkins algorithm uses a von Neumann neighborhood. Its rules are as follows: The cell will be alive in the next generation if the number of neighbors is odd. Otherwise, the cell will be dead in the next iteration of the automaton.

Saldaña and Yu [17] made use of standard CA algorithms to run on the AGILA cluster, using the data parallelization approach. Their MPI implementation of the CA rules divides the CA lattice into smaller matrices and distributes these data to  $N$  running processes. In the current study, the authors modified the C programs to include Fredkins CA rule.

## 4. RESULTS

The results were plotted fitted to an exponential function using the "fit" command of gnuplot [18].

### 4.1 HPL Benchmarking

The current peak performance of the two clusters were measured by increasing the problem size  $N$  up until the entire RAM was filled up by the dense matrix.  $N$  was obtained using simple dimensional analysis described in 2. Most of the memory will contain the  $N^2$  double precision floating point numbers which is 8 bytes in size. The HPL website [8] suggested that program should take up 80% of the cluster's volatile memory.

$$N = \sqrt{\text{Total Memory Size in bytes} \cdot \frac{1 \text{ double}}{8 \text{ bytes}}} \quad (2)$$

The AGILA HPCS reached a peak performance of 3.4 Giga-flops. This is slightly higher than the first documented peak performance of AGILA, i.e., 3.16 Gflops [2,3]. In contrast to AGILA, the Ateneo's new cluster MEDGRID, reached a peak performance of 40 Gigaflops. But it can be noticeable in Figure 4 that the distribution of HPL performance in higher problem sizes starts to scatter from problem size 20000 to 28000. MEDGRID stabilizes to about 35 Gigaflops as described into an exponential trendline.

To demonstrate the effects of the number of processors to HPL, the grid ratio  $P$  and  $Q$  are changed in the HPL.dat configuration file as described in Table 1. As expected from

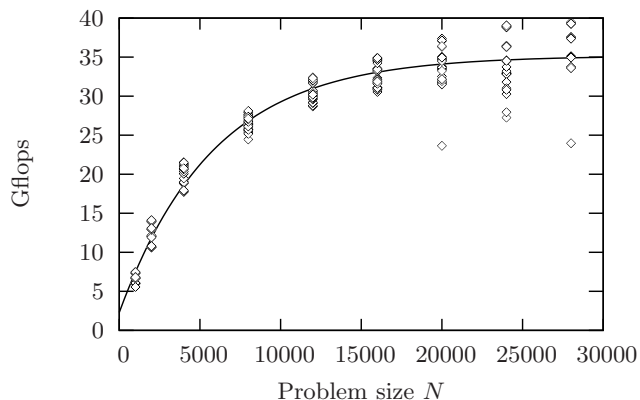


Figure 4: HPL benchmark results showing MEDGRID's peak performance using 14 processors

the previous peak performance test, MEDGRID is significantly faster than AGILA. Another interesting observation from Figure is that for every processor added to accomplish HPL, the performance contribution decreases. It exhibits a variant of the "diminishing returns principle".

### 4.2 CA Benchmarking

The timing metrics of the modified CAEngine was obtained by running a 1000x1000 grid lattice for 500 iterations. It can be observed in Figures 6 and 7 that the performance improvements in simulation time decreases as the MPI environment tries to run more processes. This behavior is the same as the timing metrics tests performed using HPL in the previous section. There is also a minimal different with the performance of the Game of Life and Fredkin's rule. The grid size on AGILA was constrained to small dimensions because it can only handle small amounts of data because of its limited memory size. The same grid size is applied to run on MEDGRID. MEDGRID executed the program 10 times faster than AGILA as seen in Figure 7. It is interesting to note, however, that running a single process on MEDGRID gave better performance than two processors.

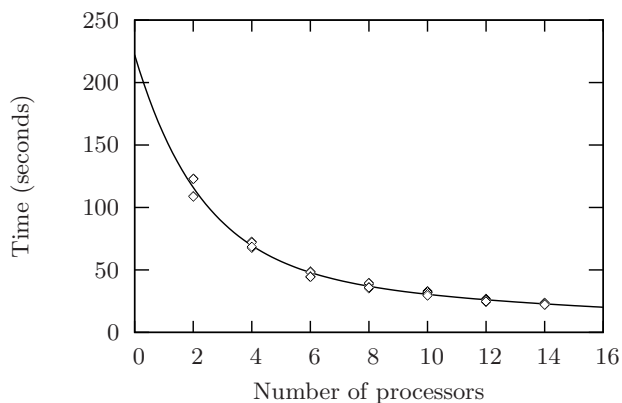


Figure 5: MEDGRID HPL performance vs the number of processors using a problem size  $N = 10000$

**Table 1: Grid ratio configuration for HPL in processor scaling**

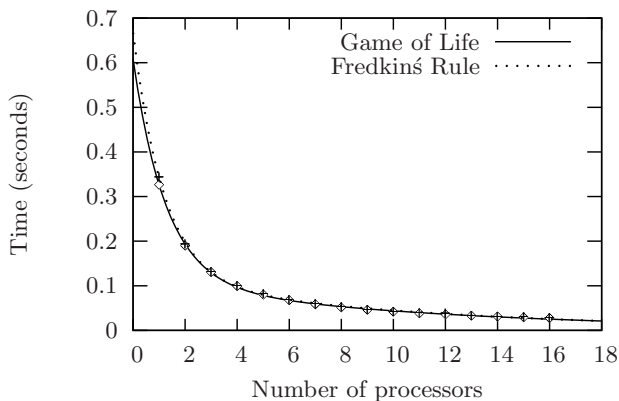
Processors	$P$	$Q$
2	1	2
4	2	2
6	2	3
8	2	4
10	2	5
12	3	4
14	2	7

It can be inferred that the properties of Symmetric Multiprocessor architectures played a role in the process. It maybe possible that the inter-process communications between the two processors in a dual CPU machine in the MEDGRID cluster created an overhead. This may have caused the delay during the event when two of the processors in a compute node tried to communicate with each other.

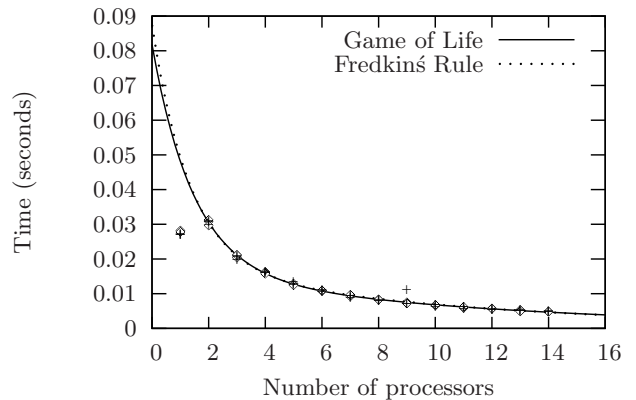
## 5. CONCLUSIONS

The MEDGRID cluster is found to be significantly faster and more efficient in performance than the AGILA cluster. This can be attributed to the fact the MEDGRID cluster is younger and is powered by Dual Xeon processors as compared to AGILA cluster which is made up of AMD K6 and Athlon processors at half the clock speed. According to the Rocks Cluster list of users [19], two CPU commodity Pentium 4 machines can achieve a peak performance of 3 Gigaflops – which is approximately the peak speed of the AGILA cluster. Therefore, it is highly recommended that AGILA cluster be prepared for decommissioning or for upgrading to a newer cluster.

Although cellular automata algorithms do not use floating point operations, benchmarking Beowulf clusters using CA is still possible, as demonstrated in this study. Both the timing metrics using HPL and CA (see Figures 6 and Figure 7) exhibited the same behavior in different timing performances.



**Figure 6: Timing performance of CA algorithms using AGILA with 16 processors**



**Figure 7: Timing performance of CA algorithms in MEDGRID with 14 processors**

## 6. REFERENCES

- [1] Beowulf.org: Overview. [Online]. Available: <http://www.beowulf.org/overview/index.html>
- [2] R. Saldana, F. Muga, J. Garcia, and W. Yu, "AGILA: The Ateneo High Performance Computing System," in *Proceedings of the First Philippine Computing Science Congress*, Manila, Philippines, Nov 2000.
- [3] J. G. R. Saldana, F. Muga and W. Yu, "Development of a Beowulf-class High Performance Computing System for Computational Science Applications," in *Science Diliman*, Manila, Philippines: University of the Philippines, 2000.
- [4] E. Bagarinao, L. Sarmenta, Y. Tanaka, K. Matsuo, and T. Nakai, "Real-Time Functional MRI Analysis Using Grid Computing," in *Proceedings of the 7th International Conference On High Performance Computing and Grid in Asia Pacific Region*, Omiya sonic City, Tokyo Area, Japan, July 2004.
- [5] Myrinet Product List. Myricom Inc. [Online]. Available: [http://www.myri.com/myrinet/product\\_list.html](http://www.myri.com/myrinet/product_list.html)
- [6] C. M. Jack Dongarra, Jim Bunch and P. Stewart. LINPACK. [Online]. Available: <http://www.netlib.org/linpack>
- [7] The Top500 Supercomputer Sites. [Online]. Available: <http://www.top500.org>
- [8] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary. HPL-A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. [Online]. Available: <http://www.netlib.org/benchmark/hpl>
- [9] HPL Algorithm. [Online]. Available: <http://www.netlib.org/benchmark/hpl/algorithm.html>
- [10] FLOPS. Wikipedia, The Free Encyclopedia. [Online]. Available: <http://en.wikipedia.org/wiki/FLOPS>
- [11] BLAS (Basic Linear Algebra Subprograms). [Online]. Available: <http://www.netlib.org/blas>
- [12] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–27.

- [13] Automatically Tuned Linear Algebra Software (ATLAS). [Online]. Available: <http://math-atlas.sourceforge.net>
- [14] PC Cluster Consortium. [Online]. Available: <http://www.pccluster.org>
- [15] W. Gropp and E. Lusk, *Users Guide for mpich, a Portable Implementation of MPI*, Argonne National Laboratory, 1994.
- [16] J. Conway, "The Game of Life," *Scientific American*, vol. 223, pp. 120–123, 1970.
- [17] R. Saldana and W. E. Yu, "Cellular automata explorations on a Beowulf cluster computer," Japan: Yokohama National University, 2001.
- [18] T. Williams and C. Kelley, *gnuplot: An Interactive Plotting Program*, 1998.
- [19] Rocks Clusters Register. [Online]. Available: <http://www.rocksclusters.org/rocks-register>