Comparison of Parallel Processing Systems

May 2004

George Mason University

Computer Science 672-Computer System Performance Evaluation

Ash Dean
Katherine Willis
“Yesterday’s supercomputer is today’s personal computer.”\(^1\) When thinking about how rapidly computers are changing, the thought of common desktop computers competing with traditional supercomputers comes to mind. With the slow corporate transition from expensive, traditional supercomputers to inexpensive Beowulf clusters, one must wonder if a supercomputer retains any performance benefit over a cluster. The purpose of this project is to determine what, if any, performance benefits a supercomputer has over a cluster; focusing on determining if there is computational speedup gained by using a traditional supercomputer versus using a Beowulf cluster when solving a dense system of linear equations. A second metric investigated in this project is the nature of the speedup curve for the Beowulf cluster when solving dense systems of equations. Analytical models of the two systems, the supercomputer and the cluster, are used to compare the systems. Due to the expense and availability of supercomputers, online documentation of traditional supercomputer performance results were used to compare results found on the Beowulf cluster at George Mason University when executing the Linpack benchmark software. The supercomputer study used for this project is the Linpack Benchmark\(^2\), which is a measure of a computer’s floating-point rate of execution. The rate of execution is determined by running a program that solves a dense system of linear equations. More specifically, we used the parallel version of the Linpack code, HPL, for our study. We used results of running HPL on supercomputers from the internet (the top500 list); we downloaded the software and executed it on the Beowulf cluster at GMU to get results to compare with the supercomputer. We hope to provide a small insight to industry for the best way to move towards the parallel computing framework.

**Metrics:**

The model of a traditional supercomputer uses a single system containing multiple processors, sometimes arranged as a load dependent device and sometimes arranged as separate load independent devices, and, perhaps multiple disks. However, the question at hand does not necessitate the modeling of I/O subsystems; this project is only concerned with the computational speedup gained by using a traditional supercomputer. The model of a Beowulf cluster is somewhat more sophisticated. Instead of modeling multiple processors inside a single system, a Beowulf cluster consists of multiple, possibly heterogeneous systems (nodes), linked together to form a larger system. For the purposes of our project, we used a Beowulf cluster at George Mason University comprised of at most eight homogeneous nodes at one time. Each node consists of an AMD Athlon XP1700+ processor that runs at approximately 1466.756 megahertz, 256 kilobytes of cache, and 256 megabytes of main memory.

The first metric used in our study is computational speedup. Speedup is defined to be the ratio between the time necessary to complete a computation in serial and the time necessary to complete the same computation in parallel. We wanted to determine if a computation speedup is gained by using a supercomputer versus a Beowulf cluster. The formula used for speedup is:

---

The formula above is derived from the process of trying to increase throughput on a
parallel system. In order to increase throughput on a parallel system, one must decrease
the overall execution time, $T_0$, required to process some $n$ jobs ($C_0$). Or alternatively, the
number of jobs ($C_0$) executed has to increase in the same time, $T_0$. Overall throughput is
measured as $X = \frac{C_0}{T_0}$. We knew the number of jobs in the system that we wanted to
execute so we were able to keep that constant and only look at the time to execute the
jobs with a varying number of processors in the supercomputer or nodes in the Beowulf
cluster.

The second metric explored is the nature of the speedup curve for the super
computer and the Beowulf cluster, also known as efficiency. This metric is used to
ascertain the benefit of each additional processor in the system. Efficiency is measured
as the ratio of speedup to the number of processors. The formula is:

$$ E_p = \frac{S_p}{p} $$

where $E_1 = S_1 = 1$

This metric is essentially a measure of overhead, as the number of processors increases,
overhead increases as well. Analytical models must be considered in order to find the
metrics described.

**Analytical Comparisons:**

**Traditional Supercomputer:**

A traditional supercomputer is viewed as one single machine with multiple, often
very fast processors (see Figure 1.). In our model of the supercomputer, the jobs in our
system have equivalent workloads and the number of jobs in the system remains constant
thus allowing us to use a closed, single class queuing network.
We used ClosedQN.xls to solve the queuing network assuming a service demand of .08 seconds and varying the number of jobs in the system. We assumed a service demand because we are simply interested in the shape of the curve of the throughput, as opposed to actual numbers, in order to compare it with the throughput curve obtained with the Beowulf cluster. The throughput versus number of processors in the system is graphed below (Figure 3) using the throughput obtained from using ClosedQN.xls along with the results from the Beowulf cluster.

Beowulf Cluster:

With a Beowulf cluster, jobs are divided into tasks that are processed concurrently on different nodes in the cluster. The tasks communicate with one another using message passing not by utilizing shared memory; this allows for independent execution and completion of the assigned task. We used a closed fork join queuing network to model the Beowulf cluster (see Figure 2.).

![Figure 2. Closed Fork-Join Network for Beowulf Cluster.](image)

We used a mean value analysis technique to solve the closed fork-join network. The basis for the technique is an equation that relates the response time of a network to the service times and the mean queue length of the system\(^3\). The equation is:

\[
R_i(m) \approx \frac{1}{\mu_i} \left[ H_{K_i} + Q_i(m-1) \right]
\]

where \(R_i\) is the response time, \(1/\mu_i\) is the service time, \(H\) is the k\(^{th}\) Harmonic number, \(K_i\) is the number of tasks a job is split into, \(Q\) is the mean number of jobs in the subsystem \(i\), and \(m\) is the multiprogramming level. With the above equation as the basis, the MVA technique requires iteratively (starting with \(Q(0) = 0\)) solving the equations:

\(^3\) Varki, Elizabeth. “Mean Value Technique for Closed Fork-Join Networks” Department of Computer Science, University of New Hampshire.
where $CT$ is the mean cycle time spent in a closed network, $R$ is response time as above, $V$ is the average number of visits per job, $m$ is the multiprogramming level, $X$ is the mean throughput, and $Q$ is the mean number of jobs in the subsystem $i$. This technique does not provide an exact solution but is a good approximation “…since it is derived from an equation that exactly relates the response time of parallel systems to the degree of parallelism and the mean arrival queue length.”

With further derivations and simplifications (including using Markov models), the response time of a closed fork-join network with homogeneous servers can be represented as:

$$R = s[H + A]$$

Where $R$ is the response time, $s$ is the mean service time, $H$ is the harmonic number and $A$ is the queue length. Using the above equation and Little’s law we were able to compute throughput values for our system. As we add more processors to our system, the queue length shrinks and thus the response time also shrinks. Again, we assumed a

![Graph](image)

Figure 3. Throughput comparison of supercomputer and Beowulf cluster.

---

service demand of 0.08 seconds, in order to accurately compare the supercomputer to the cluster, and varied the number of concurrently executing jobs. The throughput versus the number of processors is shown in Figure 3, and one can see that the overall curve for the cluster is not the same as the curve for the supercomputer.

From the graph in Figure 3, one can see that as the number of processors increases in the beginning, the throughput rapidly increases for the supercomputer and the throughput slowly increases for the Beowulf cluster. One can also notice that there is a point where additional processors make no substantial increase in throughput for both the supercomputer and the cluster. With an understanding of the analytical models, we can now explain the executions of the Linpack benchmark we completed, explain results and draw conclusions.

LINPACK Benchmark:

The original intention of the Linpack benchmark was to give users of the Linpack software an idea of how long the software would take to solve matrix problems. The Linpack package is a collection of Fortran subroutines for solving systems of linear equations. The overall idea of the package is to decompose the original problem into a product of simpler matrices that are easily manipulated and used to solve the original problem. Linpack is built on another software package called, BLAS, which is a collection of linear algebra subroutines. The first Linpack benchmark report appeared as an appendix in the first user’s guide for the Linpack software in the late 1970s.

There are three benchmarks in the Linpack Benchmark report and this project focuses primarily on the Highly Parallel Computing benchmark (HPL), which is designed for distributed memory systems. The guidelines for running the HPL benchmark allows for replacement of the LU factorization technique used in the benchmark so long as the accuracy of the solution satisfies the bound:

$$\frac{\|Ax - b\|}{\|A\|\|x\|\|e\|} \leq O(1)$$

where $\epsilon$ is the machine precision (i.e., $2^{-53}$ on IEEE computers) and $n$ is the size of the problem. For our execution of the HPL software we used the standard LU factorization technique that is included in the benchmark software. The guidelines also require that the matrix in the driver program be used during the run of the software. The matrix in the driver program is generated using a pseudo-random number generator that designs the matrices so that they are forced to use partial pivoting in Gaussian Elimination. All guidelines for the HPL benchmark were followed during our executions. We also needed implementation of the MPI and BLAS packages that were required by HPL.

The HPL software package generates and solves random dense systems of linear equations on distributed memory computers. The package allows users to choose the factorization method used and provides results to the user. In our executions of the HPL software we chose to use Gaussian Elimination for the factorization of the matrices. People often learn Gaussian Elimination of $Ax = b$, by adjoining the right hand side vector $b$ to the matrix $A$, then performing row combinations that would leave ones on the diagonal and zeros elsewhere for entries of the matrix $A$. Gaussian Elimination is often referred to as LU factorization in modern times. LU factorization works similar to the
Gaussian elimination students often learn by hand with one main difference, that is, that the vector \( b \) is ignored until after the matrix \( A \) has been processed into the product of two matrices, \( L \) and \( U \), and a matrix \( P \) is found that is a permutation of \( A \) (ones on the diagonal and zeros elsewhere). The equation now becomes \( LUx = Pb \), and since \( PAx = Pb \) only three steps are required to solve the new equation. This is the basis for the algorithm used by HPL to solve the systems of linear equations.

Supercomputer Performance:

For our analysis of supercomputer performance we used results from the Linpack benchmark that were available online with the TOP500 list. The TOP500 list can be found at [http://www.top500.org](http://www.top500.org). It is important to remember that these results cannot predict the overall performance of system; rather they predict the performance of the system when solving dense systems of linear equations. The following metrics describe what is expected to occur on a supercomputer in terms of speedup and efficiency when solving dense systems of linear equations.

**Speedup**

Speedup is defined as the ratio between the time necessary to complete a computation in serial and the time necessary to complete the computation in parallel. As mentioned above the formula for speedup is:

\[
S_p = \frac{T_1}{T_p}
\]

From our project we determined that a supercomputer with four processors has an average speedup of 3.83 and a supercomputer with eight processors has an average speedup of 5.77. The following box and whisker plots illustrate the results we obtained for speedup for an eight processor and a four-processor supercomputer.
Average speedup of an eight-processor supercomputer

Efficiency
This metric gives insight to the nature of the speedup curve due to the number of processors in a system. Efficiency is measured as a ratio of speedup to the number of processors in a system. The formula for efficiency is:

\[ E_p = \frac{S_p}{p} \]

where \( E_1 = S_1 = 1 \)

A four-processor supercomputer averages a speedup of 0.845 and an eight-processor supercomputer averages a speedup of 0.713. The following box and whisker plots show the results we obtained from studying the supercomputers on the TOP500 list.

Cluster Performance:
For our analysis of the Beowulf cluster when solving dense systems of linear equations, we used the cluster at George Mason University calculating results first with the cluster containing only two nodes, then again with four nodes and lastly with the cluster containing eight nodes. We downloaded the HPL software package and installed the software on the cluster along with an implementation of MPI, a message passing interface package, and BLAS, a package of subroutines used for linear algebra. The HPL software outputs various numbers including execution time and the number gigaflops (floating point operations) computed. Using the output from the software, we were able to compute our metrics on the Beowulf cluster. Using the same formulas for speedup and efficiency described above we obtained the results shown in Table 1.

<table>
<thead>
<tr>
<th>Uniprocessor Time</th>
<th>Number of Processors</th>
<th>Multiprocessor Time</th>
<th>Speedup</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.5</td>
<td>2</td>
<td>4.9734</td>
<td>4.926</td>
<td>2.487</td>
</tr>
<tr>
<td>24.5</td>
<td>4</td>
<td>.3396</td>
<td>72.144</td>
<td>18.036</td>
</tr>
<tr>
<td>24.5</td>
<td>8</td>
<td>.3127</td>
<td>78.349</td>
<td>9.793</td>
</tr>
</tbody>
</table>

Table 1. Beowulf cluster performance.

From the results in Table 1, the nature of the speedup curve for a cluster can be seen and it is clear that there is an enormous speedup when the number of processors in the cluster goes from two to four (Figure 4). When more nodes are added to the cluster a leveling off of speedup can be seen. One can conclude that there is a speedup benefit when using approximately four nodes in the cluster but adding many more may not significantly improve speedup on the cluster.

![Speedup vs. Number of Processors](image)
Conclusions:

From the data gathered and studied, one can assume that a Beowulf cluster is an ideal solution when only a small number of processors are necessary when solving systems of equations. Once the number of processors increases to above approximately 8, a supercomputer would be the best solution for solving dense systems of linear equations. In terms of raw computing power (at least for solving systems of linear equations), a Beowulf cluster cannot match traditional supercomputers with upwards of 128 processing elements. As for cost, building a Beowulf cluster can be significantly cheaper than buying a supercomputer and should be considered as an alternative to the supercomputer provided a relatively small number of processors are required. When the task at hand is solving dense systems of equations, one can expect a Beowulf cluster to have reasonable execution times and a more appealing cost than a traditional supercomputer.