LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-216011

Report for CS 698-95 #Directed Research # Performance Modeling:# Using Queueing Network Modeling to Analyze the University of San Francisco Keck Cluster Supercomputer

M. L. Elliott

October 10, 2005

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Report for CS 698-95
"Directed Research – Performance Modeling:"

Using Queueing Network Modeling to Analyze the University of San Francisco Keck Cluster Supercomputer

Michael L. Elliott
University of San Francisco

August 19, 2005

Table of Contents

Table of Figures

I – Introduction

In today's world, the need for computing power is becoming more pressing daily. Our need to process, analyze, and store data is quickly exceeding the capabilities of small self-contained serial machines, such as the modern desktop PC. Initially, this gap was filled by the creation of supercomputers: large-scale self-contained parallel machines. However, current markets, as well as the costs to develop and maintain such machines, are quickly making such machines a rarity, used only in highly specialized environments. A third type of machine exists, however. This relatively new type of machine, known as a cluster, is built from common, and often inexpensive, commodity self-contained desktop machines. But how well do these clustered machines work?

There have been many attempts to quantify the performance of clustered computers. One approach, Queueing Network Modeling (QNM), appears to be a potentially useful and rarely tried method of modeling such systems. QNM, which has its beginnings in the modeling of traffic patterns, has expanded, and is now used to model everything from CPU and disk services, to computer systems, to service rates in store checkout lines. This history of successful usage, as well as the correspondence of QNM components to commodity clusters, suggests that QNM can be a useful tool for both the cluster designer, interested in the best value for the cost, and the user of existing machines, interested in performance rates and time-to-solution.

So, what is QNM? Queueing Network Modeling is an approach to computer system modeling where the computer is represented as a network of queues and evaluated analytically.[1] How does this correspond to clusters? There is a neat one-to-one relationship between the components of a QNM model and a cluster. For example: A cluster is made from a combination of computational nodes and network switches. Both of these fit nicely with the QNM descriptions of service centers (delay, queueing, and load-dependent). Other examples include relationships between different classes of customers in QNM and different types of messages passed on clustered systems, and the obvious relationship between the QNM model queues and message queueing in switches and network cards. Even the parameterization of QNM components lends itself well to cluster modeling. Numbers of service centers (computational nodes and switches) is generally well known for existing systems, and can be estimated for potential systems. Number of customers in the system can be estimated based on application call traces or profiles. Timing rates and service demands, too, can be estimated based on device specifications, or through application tracing or profiling. Typical results reported include throughputs, queue lengths, and response times, all of which are important to determining how well a system is utilized.

In this research, QNM is applied to the Keck Cluster as a strong scaling problem. Strong scaling is where the size of the problem to be solved remains constant even as the number of processors allocated to the solution increases. QNM could also be applied in a weak scaling

[1] [28]

manner, meaning the problem size increases as the number of allocated processors increases, but this application is not investigated here.

Initial research was conducted, and methodology refined, at Lawrence Livermore National Laboratory (LLNL) on MCR.[2] After refinement, methods developed at LLNL were applied to the Keck Cluster at the University of San Francisco (USF).

The rest of this paper is organized into the following sections:

I. Introduction
   This introduction.

II. Summary of Work Performed
   Summary of work performed for this research, both historical and recent.

III. Description of Software and Hardware Products
   Description of the various components used to create, test, and validate the QNM model.

IV. Description of Methodology
   A description of the methods and methodologies used to create, test, and validate the QNM model.

V. Analysis and Report of Findings
   An analysis of collected data and report of the findings.

VI. Future Research
   Areas of future research.

VII. Conclusion
   Conclusions and discussion.

VIII. Appendices
   Source code and other supporting documents.

[2] Development of methods done under Research Subcontract B546340 for Queuing Network Models of Performance of High End Computing Systems.

II – Summary of Work Performed

- Downloaded, installed, compiled, and ran NAS Parallel Benchmarks on the Keck Cluster at USF.
    - Source and script files modified to meet specific needs of Keck Cluster.
    - Script files created to batch execute benchmarks.

- Downloaded, installed, compiled, and linked mpiP with NAS-PB at USF.
    - Linkage for C and Fortran MPI compilers modified to include mpiP libraries.

- Executed NAS-PB with mpiP on Keck Cluster.
    - Class ranges A – D.
    - Processor ranges 1 – 64.
    - Cyclic and linear allocations.

- Modified and populated Excel spreadsheet to hold resultant values and calculate model inputs.
    - Performs error analysis and data validation.
    - Graphically displays results.
    - Initially developed as part of the LLNL Research Subcontract.

- Modified QNM software, which was developed as part of the LLNL Research Subcontract.
    - Software modified to include single class load dependent service center analysis.
    - Development near completion for multiple class load dependent service center analysis.

- Modeled Keck Cluster using modified QNM software, and used benchmarks and machine models from LLNL to help refine modeling technique.
    - Used similar techniques to Keck Cluster modeling.

- Used software, developed as part of the LLNL Research Subcontract, to create input files for QNM modeling program.
    - Fed input file to solver using indirection.

- Used software, as part of the LLNL Research Subcontract, to extract values from mpiP and NAS-PB result files for Excel spreadsheet, and applied to Keck Cluster data.
    - Delimited text file.

- Created a research bibliography, in conjunction with LLNL Research Subcontract.

III – Description of Software and Hardware Products

inMaker

A simple Java program, developed as part of the LLNL Research Subcontract, which, when given parameters for the QNM solversolver, will create as output a file that can be used as input for the QNM solver.

Keck Cluster

The University of San Francisco Department of Computer Science's supercomputer. As described on its website, the Keck Cluster is "… a 64 node Beowulf cluster … [containing] Dual Pentium III 1GHz CPUs, 1GB RAM, [and a] Myrinet Network card … connected by … a 2Gbps Myrinet network used exclusively for communication between MPI programs."[3]

The default MPI environment on the Keck Cluster is Myrinet's MPICH-GM v. 1.2.4..8a, which was used for all compilation, linkage, and execution.[4]

The Keck Cluster is a login/logout system, has no batch execution control, and runs RedHat Linux 8.0.

MCR

A multiple node supercomputer located at Lawrence Livermore National Laboratory. MCR stands for Multiprogrammatic Capability Cluster.

MCR is "a large (11.2 TF) tightly coupled Linux cluster … has 1,152 nodes, each with two 2.4GHz Pentium 4 Xeon processors and 4GB of memory … runs the LLNL CHAOS software … which incorporates … Red Hat Linux."[5]

Compilation, linkage, and execution performed on MCR using Intel compilers v. 8.1.

MCR uses a Quadrics QsNet Elan 3 interconnect, which delivers high bandwidth (>300 MB/s) with low latency (<5.0 μs).[6]

---

[3] Keck Cluster [USF-CS]. 27 Jul 2004. U. of San Francisco. 8 Aug 2005. <http://kc.cs.usfca.edu/index.shtml>.
[4] Keck Cluster [USF-CS]. 25 Jun 2004. U. of San Francisco. 8 Aug 2005. <http://kc.cs.usfca.edu/software.shtml>.
[5] M&IC Capability Cluster (MCR). 6 Aug 2004. Lawrence Livermore National Laboratory. 8 Aug 2005. <http://www.llnl.gov/linux/mcr/mcr.html>.
[6] M&IC Capability Cluster (MCR). 29 Aug, 2002. Lawrence Livermaore National Laboratory. 12 Aug, 2005. <http://www.llnl.gov/linux/mcr/background/mcr_background.html#sec212>.

mpiP

As described in its documentation: "mpiP is a lightweight profiling library for MPI applications."[7] It was developed by LLNL staff, and is a publicly available resource.

MpiP replaces an application's linkage to MPI programs, thus allowing it to collect information concerning a variety of MPI calls. After calling, the mpiP routines collect some system state data, and then call the related MPI routines. It can be linked at run-time, thus avoiding the need to recompile. Since statistics are generated only at the end of execution, mpiP has very low overhead.

mpiPfilter

A simple Java program, developed as part of the LLNL Research Subcontract, that filters the output files from mpiP and NAS-PB and creates a text file usable as input for the NBP Spreadsheet .

NAS Parallel Benchmarks

The Numerical Aerodynamic Simulation is described as: "[A] small set of programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications."[8]

The flavor used to benchmark the Keck Cluster is NPB 2.4.

Developed at NASA's Ames Research center, the benchmarks consist of two major components: five parallel kernel benchmarks and three simulated application benchmarks,[9] described as follows:

Kernel Benchmarks:

EP Embarrassingly Parallel: Compute bound with virtually no interprocessor communication.

MG Multigrid: Tests both short and long distance communication.

CG Conjugate Gradient: Tests irregular long distance communication.

FT Fast Fourier Transform: Rigorous long-distance communication test.

---

[7] mpiP: Lighweight, Scalable MPI Profiling. 29 Apr 2005. Lawrence Livermore National Laboratory. 8 Aug 2005. <http://www.llnl.gov/CASC/mpip/>.
[8] The NAS Parallel Benchmarks. 13 Oct 2004. National Aeronautics and Space Administration Advanced Supercomputing Division. 8 Aug 2005. <http://www.nas.nasa.gov/Software/NPB/>.
[9] [2]

IS Integer sort: Tests both computation speed and communication performance.

Simulated Applications:

LU Regular-sparse, block lower and upper triangular system solution. Limited parallelism.

SP Pentadiagional Solver.

BT Block tridiagonal solver.

The benchmarks are configurable, at compile time, for multiple machine and class sizes. Class groups (A, B, C, and D) provide increasingly larger problems used to test MPI. Certain tests have restrictions on the problem sizes (BT and SP must be n^2, CG, FT, IS, LU and MG must be 2^n, where n is the number of processors. There are no size restrictions for EP.), For simplicity, the test sizes on the Keck cluster were restricted to 2^{2n}.

NPB Spreadsheet

The NPB spreadsheet, developed as part of the LLNL Research Subcontract, is designed to receive, as input, select values from the NAS suite and mpiP files, as generated by mpiPfilter. It then uses these values to calculate model inputs for the QNM SolverSolver.

The spreadsheet also performs error analysis between modeled and measured values, and graphically displays the results with a breakdown of the components that were used to make the model, and their resultant model inputs.

It also graphically compares the components of the model's wall clock time for the application to the measured components of the wall clock time.

QNM Solver

The QNM (Queueing Network Model) Solver is a Java program, ported from algorithms and FORTRAN code in [28], and developed as part of the LLNL Research Subcontract. Using input files generated by inMaker, based on values from the NPB spreadsheet, gathered from mpiP and NAS PB suite files, the program models the system as a queueing network. The solver performs single and multiple class mean value analysis, single class load dependent service center modeling, and is capable of batch execution.

The output of the solver is entered into the NPB spreadsheet to complete the modeled vs. measured validations.

IV – Description of Methodology

Collection and Analysis Procedure

Using procedures refined using MCR at LLNL; the following methods were applied to the collection and analyzing of data from the Keck Cluster:

- The NAS Parallel benchmarks were downloaded from the NAS website.[10]
- Configuration files for the NAS PB were modified for the Keck Cluster. Minor errors in benchmark code were corrected.
- Downloaded, compiled, and installed mpiP from LLNL website.[11]
- The NAS PB executables were compiled using mpiP linkage.
- Shell scripts were written for each suite of NAS PB tests to provide proper environment setup and to ease execution.
- The shell scripts were executed, and the resultant data files were captured and stored in a directory.
- The mpiPfilter program was run on the data. The resultant output files were also stored and imported into the NPB spreadsheet.
- Calculations in the NPB spreadsheet produced inputs for the QNM solver.
- QNM solver input files were created using inMaker, and fed to the solver using single class MVA batch mode.
- Outputs from the QNM solver were copied into the NBP spreadsheet, which performed error analysis on the modeled vs. measured inputs and graphically displayed the results.
- The NPB spreadsheet was then inspected for continuity, alignment of data, error, and other anomalies by the researcher.
- As necessary, the model was refined and the QNM solver was rerun on the data.

[10] http://www.nas.nasa.gov/Software/NPB/
[11] http://www.llnl.gov/CASC/mpip/

QNM Solver Input Calculation

The NPB spreadsheet, in addition to providing a common location for collected data, assists in automating the process of producing input values for the QNM solver. These values are determined as follows:

- Number of Customers (N): The number of customers is equal to the number of processors (P) allocated to the problem. This models a system in which P messages are simultaneously circulated within the system, one message for each processor.

$$N = P$$

- Number of Centers (K): This is equal to the number of processors plus two. This allows for modeling the network switch and processor computation as separate centers, and eases input creation.

$$K = P + 2$$

- Switch Delay (D_0): Switch delay models the network interconnect and is determined by dividing average message size (L) by bandwidth (BW) and adding network latency (Lat). It represents the average network transfer time for a message.

$$D_0 = \frac{L}{BW} + Lat$$

- CPU Service Demand (D_k): This value represents the amount of time per message spent by each processor when servicing MPI calls, but excludes time spent waiting. It equals the MPI time minus the MPI wait time, all divided by the product of the number of processors and number of messages (M).

$$D_k = \frac{MPI\_Time - MPI\_Wait}{P \bullet M}$$

- Computation Delay (D_{P+1}): The amount of non-overlapped per message computation between MPI calls. It is the difference between application time and MPI time, all divided by M. It represents an aggregate amount of computation across all processors.

$$D_{P+1} = \frac{App\_Time - MPI\_Time}{M}$$

NPB Spreadsheet Spreadsheet Error Analysis and Display

As described below, the NPB spreadsheet performs error analysis on modeled vs. measured values for the QNM method. Application times and graphical views are also interpreted visually, in graphs.

Validation Views

Application (Wall Clock) Time

- Observed Application Time (AT^*): $AT^* = \frac{App\_Time}{P}$
- Model Application Time (AT): $AT = \frac{R \bullet M}{P}$
- Relative Error (E_{AT}): $E_{AT} = \frac{AT - AT^*}{AT^*}$

MPI Active Time

- Observed MPI Time (MT^*): $MT^* = \frac{MPI\_Time}{P}$
- Model Application Time (MT): $MT = (R_k \bullet M) + \frac{R_0 \bullet M}{P}$
- Relative Error (E_{MT}): $E_{MT} = \frac{MT - MT^*}{MT^*}$

MPI Wait Time

- Observed Application Time (WT^*): $WT^* = \frac{MPI\_Wait}{P}$
- Model Application Time (WT): $WT = M(R_K - D_K) + \frac{R_0 \bullet M}{P}$
- Relative Error (E_{WT}): $E_{WT} = \frac{WT - WT^*}{WT^*}$

Throughput

- Observed Application Time (X^*): $X^* = \frac{M}{AT^*}$
- Model Application Time (X): X
- Relative Error (E_X): $E_X = \frac{X - X^*}{X^*}$

Graphical Views

Measured Component Time

Components of measured wall clock time.

- MPI Wait: Measured value for the aggregate time spent waiting on MPI calls.
- MPI Active: Measured value for the aggregate time spent actively on MPI calls. Also known as MPI service time.
- Computation: Measured value for the aggregate time spent in non-overlapped computation (not processing MPI calls or waiting on them).

Modeled Component Time

Components of modeled wall clock time.

- Switch Delay: Calculated amount of time it takes the network switch to process all the messages. Same as aggregate message transfer time for all messages.

  $$SwitchDelay = R_0 \bullet M$$

- MPI Contention: Calculated amount of aggregate time MPI waits for servicing of all messages..

  $$MPIContention = (R_k - D_k)M \bullet P$$

- MPI Active: Calculated amount of aggregate time MPI actively services messages.

  $$MPIActive = D_k \bullet M \bullet P$$

- Compute Time: Calculated amount of aggregate time spent on non-overlapped computation, not related to MPI.

  $$ComputeTime = MsgCompute \bullet M$$

V – Analysis and Report of Findings

CG A Analysis

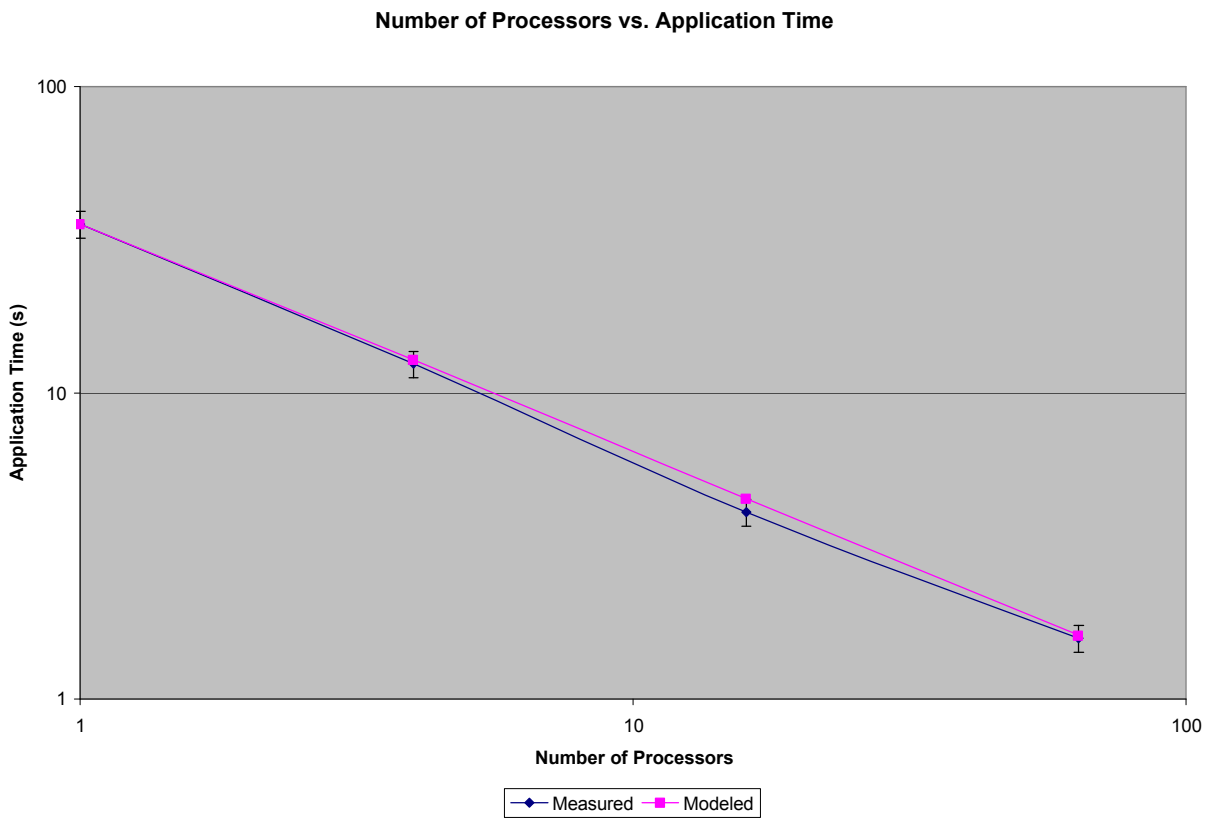

Figure 1 – CG A Wall Clock Times

As seen in Figure 1, there is good correlation between measured and modeled values for application execution time, or wall clock time (WCT). All relative error values fall within the 30% tolerance typical typical of QNM models, and most fall within 10%. The only value to fall outside 10% is the result for 16 processors, with a relative error of 10.6%

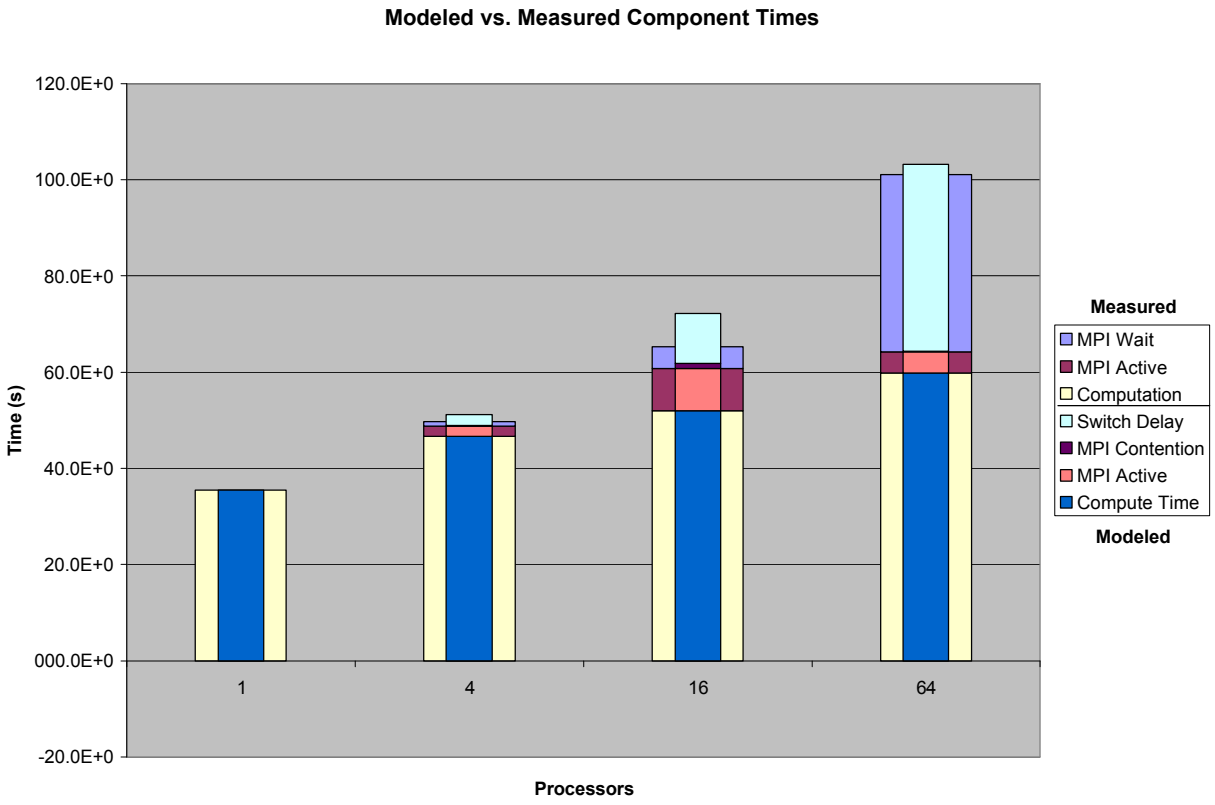

Figure 2 – CG A Component Values

Again, good correlation is seen between measured and modeled values. Also, again, discrepancies are noted in the 16 processor model, showing that refinemente of switch delay ant/or MPI wait time is in order.

CG B Analysis

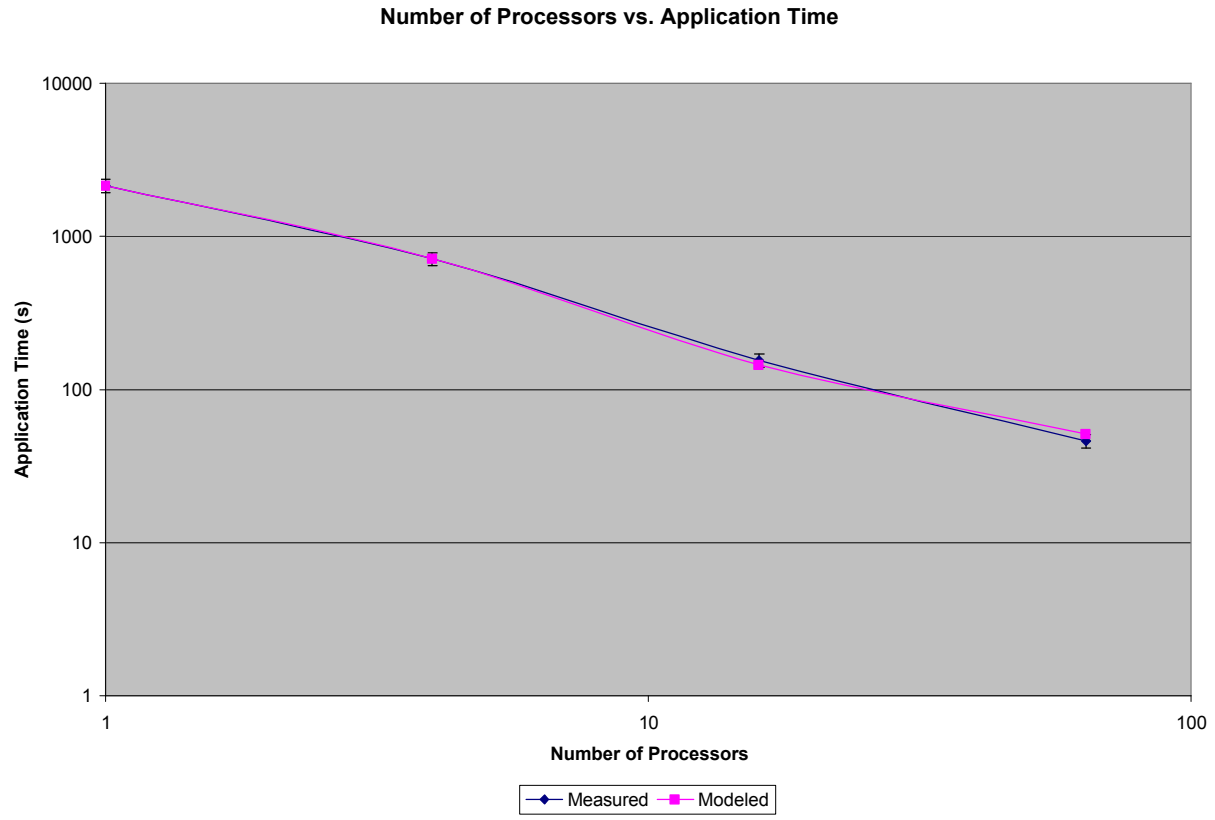

Figure 3 – CG B Wall Clock Times

Good correlation between measured and modeled values. All values fall within 10% except for 64 processors, which falls at 11.35%

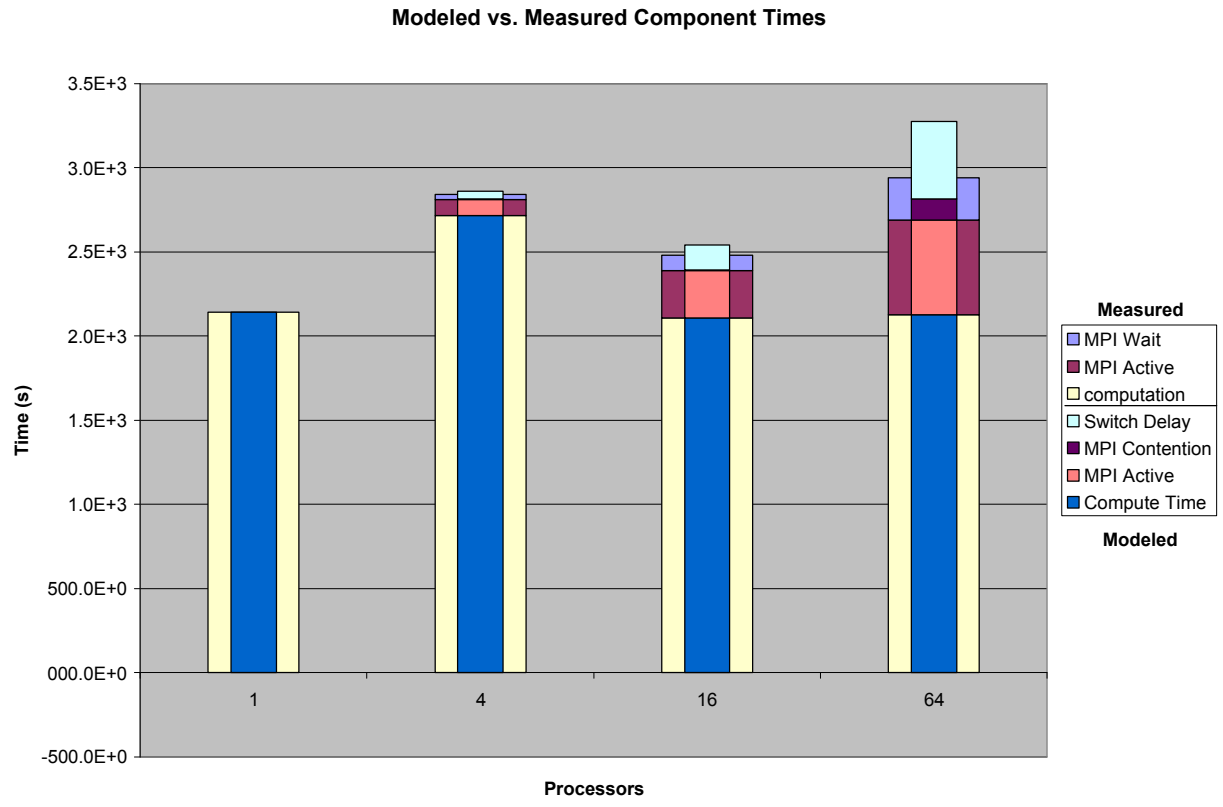

Figure 4 – CG B Component Values

Good relationship again noted for all values except 64 processors, which contains unexpectedinappropriate values for either MPI Wait or switch delay, and thus a need for further refinement of these values.

CG C Analysis

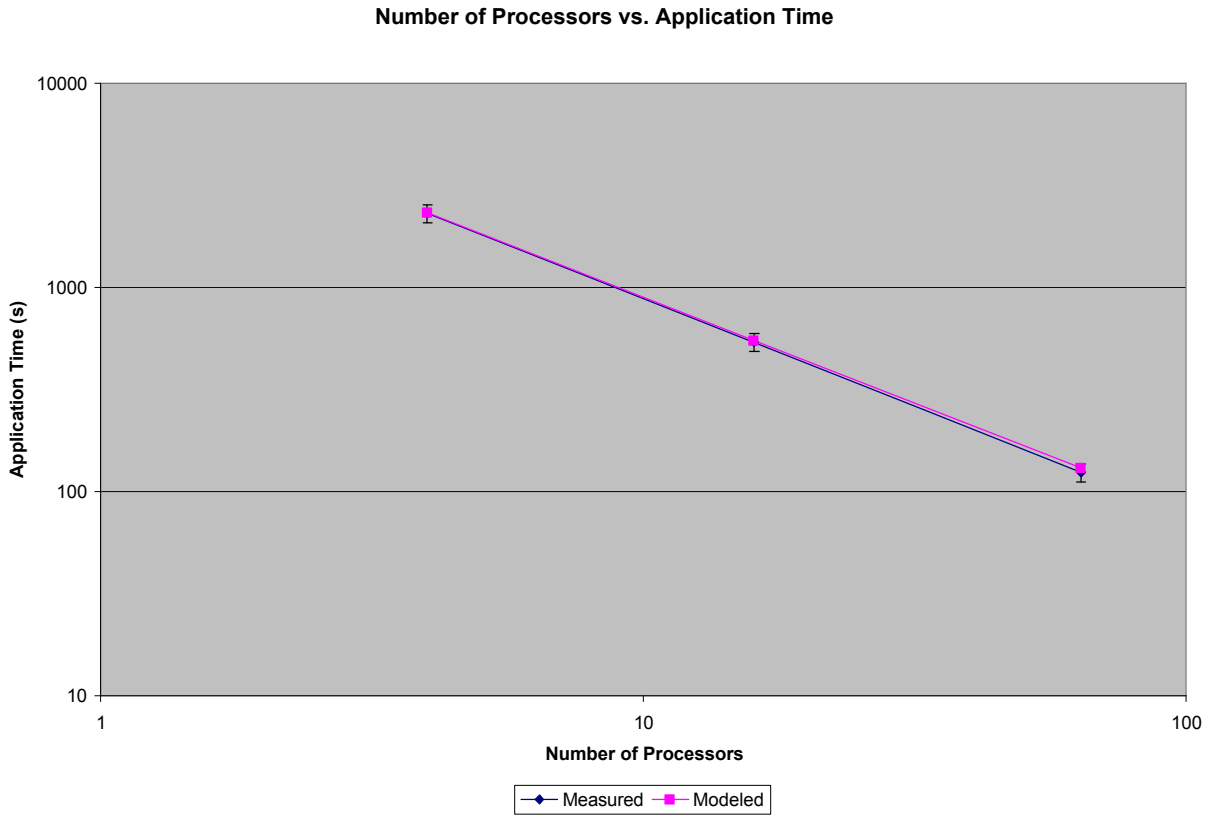

Figure 5 – CG C Wall Clock Times

Excellent relationship between measured and modeled values, with all values falling within 10% tolerance.

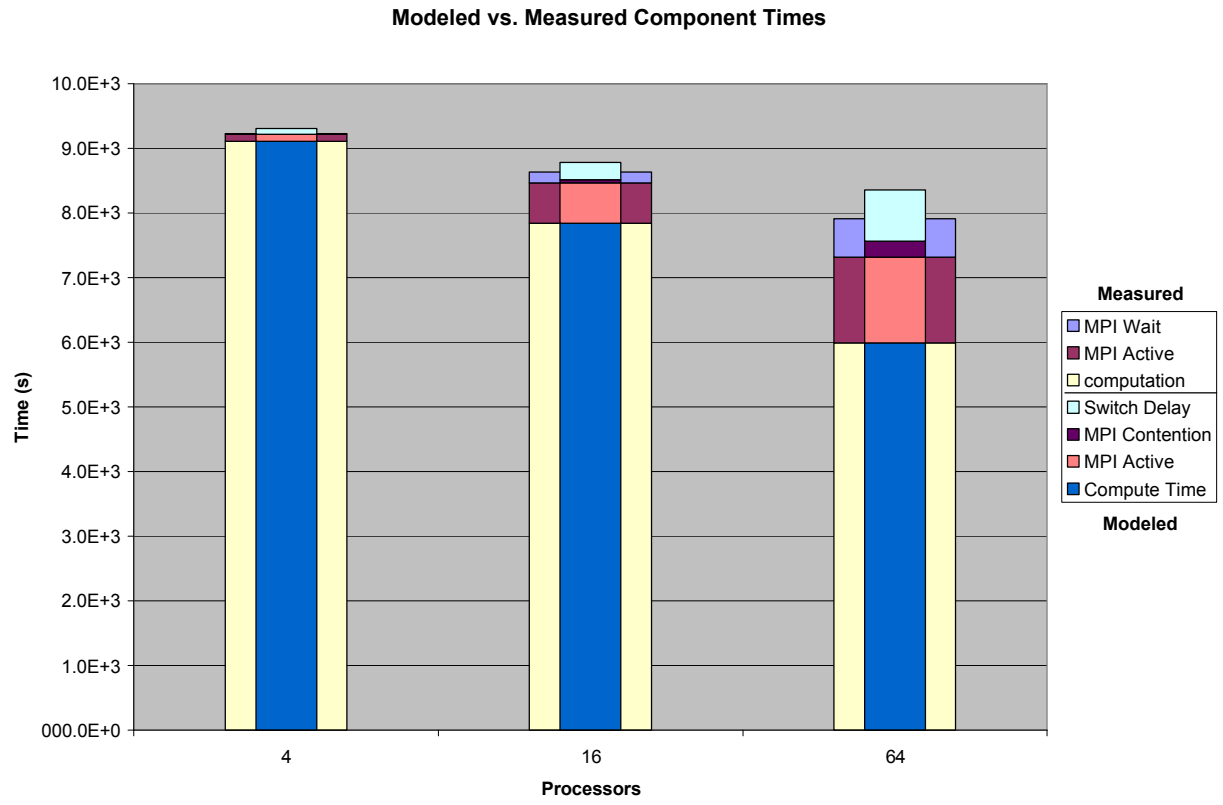

Figure 6 – CG C Component Values

Good relationship between measured and modeled values. Sixty-four processor values again off for MPI Wait and switch delay.

CG D Analysis

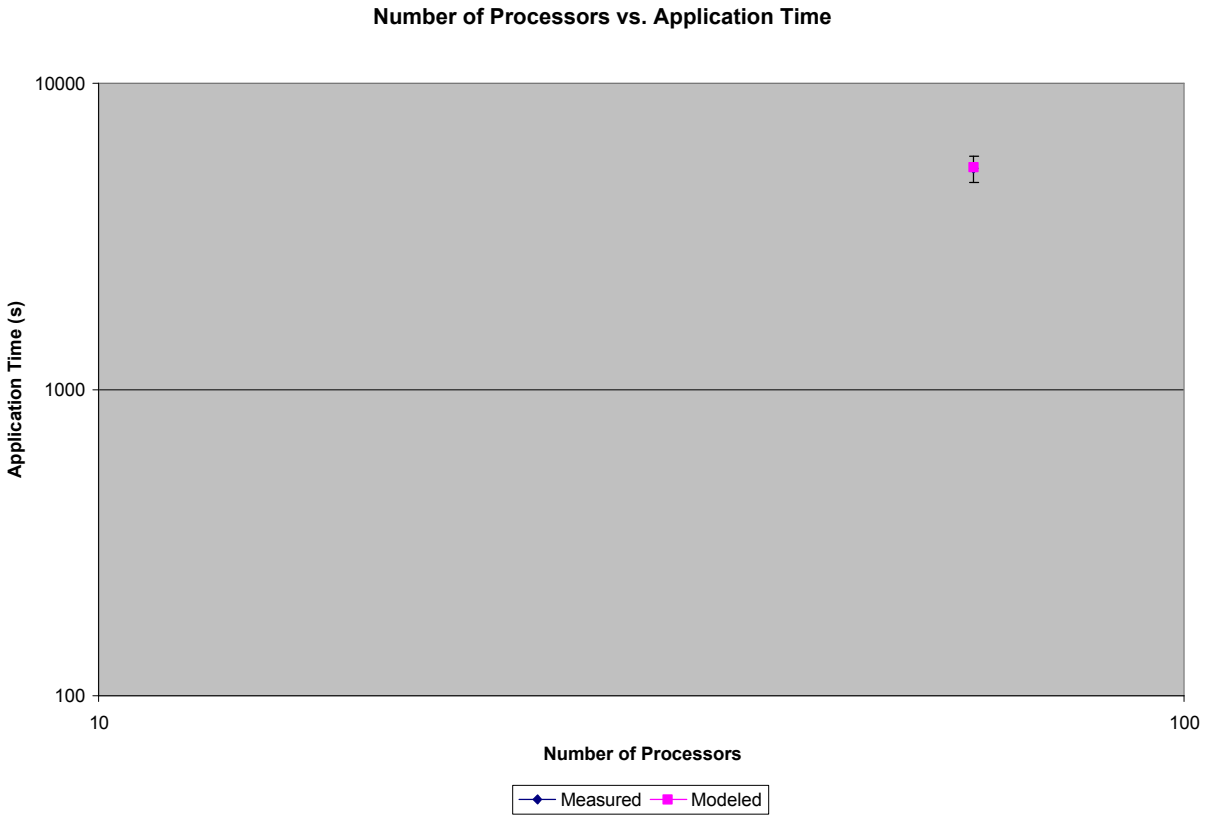

Figure 7 – CG D Wall Clock Times

For the single collected data point, there is an excellent relationship between measured and modeled values, falling at less than 10% difference.

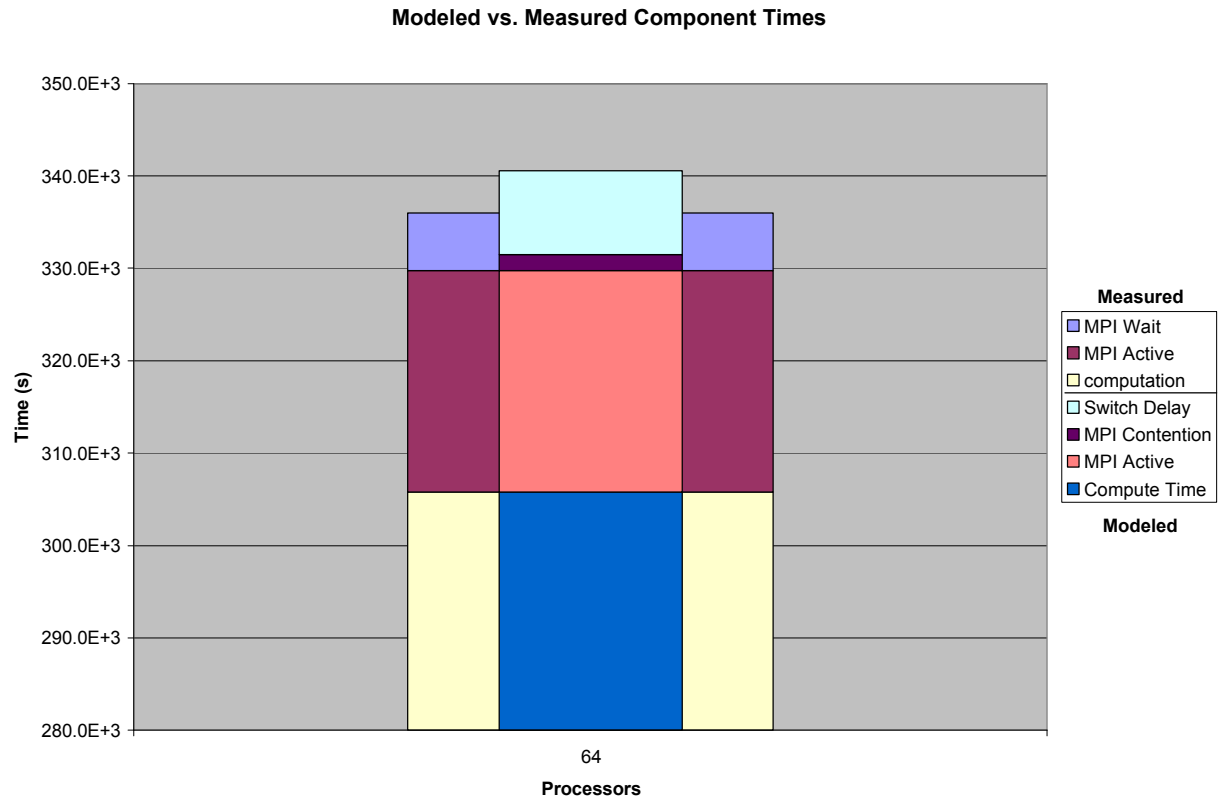

Figure 7 – CG D Component Values

Good relationship seen between measured and modeled data, however, again, the difference between measured and modeled MPI Wait and switch delay is notable.

Keck Cluster Analysis and Aggregate Results

As seen by the above graphs, not only does the QNM method function well within regard to the Keck Cluster, but the cluster also lends itself well to QNM. The comparison between modeled values and measured runt imes for the entire Keck Cluster are very impressive,and the results are highly usable, not only forin determining how well the cluster functions, but also for predicting future performance of applications on the cluster.

Analysis of the Keck cluster was not without problems, however. As is typical of any machine this size, numerous issues arose during analysis that had to be resolved, either by solving the problem, or finding a suitable workaround. Below, I outline some of the major difficulties in the Keck Cluster analysis, in order of their significance:

- Size: The most significant problem with the Keck Cluster is its relatively small size, compared to many other supercomputers. The small size makes it difficult to establish good baseline readings for the cluster, as many of the behaviors exhibited by

large MPI programs are not visible. For example: Typically, MPI programs reach a point where adding more processors to the solution actually increases time to solution. This 'U' shape is not visible on Keck Cluster models, as the cluster does not have enough nodes available to create a processing group this large. The only solution to this would be to increase the number of nodes available to the machine. The financial outlook for such an endeavor is bleak, however, and thus highly unlikely. The Keck Cluster is not intended to be a large, production supercomputer, but rather a small academic computer for personal, academic research, and will most likely remain so for the foreseeable future.

- Scheduling: The Keck Cluster uses a login/logout style allocation system. This leaves the system with a large amount of wasted compute time, as nodes are often unallocated, or even worse, allocated and idle. In short, the Keck Cluster is highly inefficient in the use of compute time. The addition of a batch scheduling system, with limited interactive login restricted to short batch jobs and small job debugging, would eliminate this waste and increase system efficiency.

- Prioritization: The Keck Cluster is non-prioritized. It makes no distinction between users; all are treated equally. This makes it possible for non-Computer Science departments to utilize large amounts of the cluster's resources, often leaving little or no compute power available for CS use. It also means that non-academic pursuits can choke out vital academic pursuits in the same manner. Implementation of a job prioritization policy would alleviate this. Possibilities include: faculty over students, CS over non-CS, and active class work over active research over non-academic work. To prevent job starvation, priority could be gradually increased over time for queued jobs, insuring they would eventually run.

- Stability: Any clustered supercomputer is, by its nature, an unstable system. But because of its relatively small size, stability on the Keck Cluster is a much larger problem. Because the cluster only contains 64 nodes with 128 processors total, for each node taken down for problems, approximately 1.6% of the total compute power is removed from the system. Thus, if the node has seven nodes down on average, more than 10% of the system is unusable at any given time due to instability. At the time of this writing, eight nodes of the cluster are out of service, for a compute loss of 16 processors and 12.5%.[12] Discussions with the system administrator indicate this instability is a product of the cluster's operating system. There are at least two possible solutions: locate and correct the unstable elements of the current operating system, or replace the operating system with one designed for supercomputing and known to be reliable.

- System Administration: The Keck Cluster lacks a dedicated system administration, instead relying on the CS department's general system administration. This means that the system administrator often lacks the resources and/or time to effectively repair malfunctions in the cluster. This results in further instability of the system, and

[12] Keck Cluster nodes: 9, 25, 27, 28, 45, 46, 62, and 64.

further increases the effects noted above. Solutions include dedicated administration for the Keck Cluster. While hiring a full-time professional staff member to maintain the cluster is economically unfeasible, these services could be performed by a part-time student assistant, and would be an enormous learning opportunity, as well as a strong professional reference.

- Software: Adapting mpiP and default Keck Cluster software packages to work together was a bit challenging, as the Keck documentation was difficult to find, less than adequate, and hard to decipher. Addition of well written tutorials covering installed software on the cluster would be a useful addition. This would also make an excellent opportunity for a part-time student assistant.

VI – Future Research

Following are areas of continuing interest, and candidates for future research:

- Complete analysis of FT benchmarks: Complete the determination of the parameters for the FT set of benchmarks and model the results. This will allow modeling of programs where MPI_Wait() is not called, but MPI spends time watingidle, none the less.

- Analyze, model, and predict BT, EP, IS, LU, MG, and SP benchmarks: Only one quarter of available data collected has been analyzed. Use the remaining data, based on runs with a small number of processorsshort run analysis, to perform prediction to large-scale machine usage. This gives strong credence to model accuracy.

- Analyze benchmark stability over numerous runs: Early analysis shows a large variability in different runs of the same class size and number of processors. Reason dictates that there should be no (or very little) variability, and graphical plots of various measurement values should be relativelypresent graphically as flat. Understanding this phenomenon would assist in creating more accurate models.

- Predict performance of a large, unknown machine based on small-run data and hardware information: Using data collected from small runs of a problem (small numbers of processors) and technical information about the hardware configuration, predict the performance of the machine on the same problem on a large scale. This again assists in validating the model.

- Predict performance of a machine yet-to-be-built: Use projected data on software and hardware performance to model a machine that is not yet operational. This is the ultimate test of model validity.

- Use QNM to predict performance for the weak scaling problem: Using a weakly scaled problem set, collect data on machine performance and analyze the effectiveness of QNM to predict weakly scaled problems.

- Model the computing power of the entire internet: Do a little research, make a good guess, and try to model the total computing power of every machine connected to the internet. A bit fanciful, maybe, but most likely very interesting.

VII – Conclusion

Based on the results above for the Keck Cluster, Queueing Network Modeling is an excellent predictor of machine performance, providing results that are were highly accurate, and often providing insight into the operation of the cluster.surprising. Despite its flaws, the Keck Cluster provides a respectable test platform, and produces highly predictable results with minimal extra work or overhead. Preliminary results from FT run analysis indicates results just as promising.

While the model does seem to currently have deficiencies when modeling some of the individual components, such as MPI Wait, that make up an MPI program, the results from wall clock time analysis, which are by far more interesting and useful, are well within the model's usual 30% error performance, and most are within scientific tolerance of 10%. Thus QNM provides meaningful results, and is useful for determining the approximate run times for large-scale, time critical applications. QNM is one solution to the strong scaling problem described in the introduction.

Another potential application of QNM would be job scheduling in large-scale batch systems. Using QNM run time approximations, the scheduler would be able to schedule more jobs, obtaining a higher system throughput, shorter time to solution, and better use of compute time. It could also be used to assist in job priority determination, to keep as many nodes in a system active as possible.

The operating system, network traffic and resource usage by other jobs, and similar noise do have the potential to create inaccuracies in measured system data, and thus can create inaccurate models. Preliminary run stability analysis shows a large amount of variability in measured values for small benchmark runs. However, it appears that QNM tends to flatten this variabilitythese differences, resulting in models that, graphically, appear much straighter and smoother.more horizontal. Reasons for this are currently under investigation, and the use of data averaged from multiple short benchmarks to filter noise is being explored.

In summary, QNM shows good potential for machine performance prediction. It gives much better accuracy than often seen in other applications, is easy to calculate, and requires little data collection. The academic, commercial, and administrative potential for QNM is large, and if current trends hold, will continue to grow. Also, since QNM is little researched in regard to computer performance modeling, the research potential for QNM is tremendous.

VIII – Appendices: Source Code and Supporting Documents

Appendix A – Bibliography

[1] Alexandrov, Albert, et al. "LogGP: Incorporating Long Messages into the LogP Model – One step closer towards a realistic model for parallel computation." Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures. 24 – 26 Jun. 1995. Santa Barbara: U. of California at Santa Barbara, 1995. 95 – 105.

[2] Bailey, D., et al. "The NAS Parallel Benchmarks." RNR Technical Report. NASA Ames Research Center, Mar. 1994.

[3] Bailey, David H., et al. "Performance Technologies for Peta-Scale Systems: A White Paper Prepared by the Performance Evaluation Research Center and Collaborators." White paper. Lawrence Berkeley National Laboratories, 2003.

[4] Balsa, André. Linux Benchmarking – Concepts. "3. FPU tests: Whetstone and Sons, Ltd." 21 Sept. 1997. The Linux Gazette. 26 Nov. 2004. <http://www.tux.org/~balsa/linux/benchmarking/articles/html/Article1d-3.html>.

[5] Barney, Blaise M. "Introduction to Parallel Computing." 21 Jun. 2004. Lawrence Livermore National Laboratory. 28 Feb. 2005. <http://www.llnl.gov/computing/tutorials/parallel_comp/>.

[6] Barney, Blaise M. "Message Passing Interface (MPI)." 21 Dec. 2004. Lawrence Livermore National Laboratory. 28 Feb. 2005. <http://www.llnl.gov/computing/tutorials/mpi/>.

[7] Barney, Blaise M. "MPI Performance Topics." 24 May 2004. Lawrence Livermore National Laboratory. 28 Feb. 2005. <http://www.llnl.gov/computing/tutorials/mpi_performance/>.

[8] Barney, Blaise M. "Performance Analysis Tools." 15 Jul 2004. Lawrence Livermore National Laboratory. 30 Mar. 2005. <http://www.llnl.gov/computing/tutorials/performance_tools/>.

[9] Bramer, Brian. System Benchmarks. DeMontfort University, UK. 26 Nov. 2004. <http://www.cse.dmu.ac.uk/~bb/Teaching/ComputerSystems/SystemBenchmarks/BenchMarks.html>.

[10] Calzarossa, Maria, Louisa Massari, and Daniele Tessera. "Workload Characterization Issues and Methodologies." Performance Evaluation, LNCS. Ed. G. Haring et al. Berlin: Springer-Verlag, 2000. 459 – 482. 9 Jul. 2004. <http://www.cs.huji.ac.il/course/2003/perf/calza1.pdf>.

[11] Cameron, Kirk and Rong Ge. "Predicting and Evaluating Distributed Communication Performance." SC 04 Proceedings. 6 – 16 Nov. 2004. Pittsburgh: David L. Lawrence Convention Center, 2004. 1 – 15.

[12] Cameron, Kirk and Vimi S. Carol. "High-Performance, Power-Aware Computing." Lecture. Lawrence Livermore National Laboratory Institute for Scientific Computing Research. Livermore, CA, May 9, 2005.

[13] Campbell, Roy L., Jr., and Larry P. Davis. "Influence of Workload Characterization on DoD High Performance Computing Modernization Program Acquisitions." Proceedings of the Second workshop on Productivity and Performance in High-End Computing. 13 Feb. 2005. San Francisco: Palace Hotel, Eleventh International Symposium on High Performance Computer Architecture, 2005. 7 – 10.

[14] Carrington, Laura, et al. "Applying an Automated Framework to Produce Accurate Blind Performance Predictions of Full-Scale HPC Applications." Proceedings: UGC 2004. 7 – 11 Jun. 2004. Williamsburg: HPCMP Office, Arlington, VA, 2004.

[15] Compaq Extended Math Library. "LAPACK." Compaq / Hewlett-Packard Inc. 9 Dec. 2004. <http://h18000.www1.hp.com/math/documentation/cxml/lapack.3dxml.html>.

[16] Culler, David, et al. "LogP: Towards a Realistic Model of Parallel Computation." Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 19 – 22 May, 1993. San Diego: SIGPLAN: ACM, 1993. 1 – 12.

[17] Deelman, Ewa, et al. "POEMS: End-to-end Performance design of Large Parallel Adaptive Computational Systems." University of Texas. 9 Jul. 2004. <http://www.cs.utexas.edu/users/poems/Papers/Wosp/wosp_dave.html>.

[18] Denning, Peter J., and Jeffrey P. Buzen. "The Operational Analysis of Queueing Network Models." ACM Computing Surveys. Vol. 10.3. New York: ACM Press, Sep. 1978. 225 – 261.

[19] Dhrystone. "What is Dhrystone? – A Word Definition from the Webopedia Computer Dictionary." Webopedia.com. 26 Nov. 2004. <http://www.webopedia.com/TERM/D/Dhrystone.html>.

[20] Dickens, Phillip M., Philip Heidelberger, and David M. Nicol. "Parallelized Direct Execution Simulation of Message-Passing Parallel Programs." 10 Jun. 1994. NASA. 9 Jun. 2004. <http://techreports.larc.nasa.gov/icase/1994/icase-1994-50.pdf>.

[21] Dongarra, Jack. Netlib Repository at UTK and ORNL. "Frequently Asked Questions on the Linpack Benchmark and Top500." 28 Oct. 2004. AT&T Bell Laboratories, et al. 26 Nov. 2004. <http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html#_Toc27885709>.

[22] Dongarra, Jack. Netlib Repository at UTK and ORNL. "LAPACK -- Linear Algebra PACKage." 28 Oct. 2004. AT&T Bell Laboratories, et al. 26 Nov. 2004. <http://www.netlib.org/lapack/>.

[23] Dongarra, Jack. Netlib Repository at UTK and ORNL. "Linpack." 28 Oct. 2004. AT&T Bell Laboratories, et al. 26 Nov. 2004. <http://www.netlib.org/linpack/>.

[24] Dongarra, Jack. "An Overview of High Performance Computing and Self-Adapting Numerical Software." Lecture. Lawrence Livermore National Laboratory ASC Institute for Terascale Simulation. Livermore, CA, May 17, 2005.

[25] Frank, Matthew I., Anant Agarwal, and Mary K. Vernon. "LoPC: Modeling Contention in Parallel Algorithms." Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 18 – 21 Jun. 1997. Las Vegas: SIGPLAN: ACM, Las Vegas. 276 – 287.

[26] Fowler, Robert, et al. "Practical Performance Measurement and Analysis with HPCToolkit." Seminar. Lawrence Livermore National Laboratory Institute for Scientific Computing Research. Livermore, CA, May, 2005.

[27] Karatza, Helen D. "Current Trends in Modelling and Simulation of Parallel and Distributed Systems. [sic]" Editorial. I. J. of Simulation, Vol. 3.1-2. 9 July 2004. <http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-3/No%201&2%20Special%20issue%20Karatza/Karatza/Karatza.pdf>.

[28] Lazowska, Edward D., et al. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[29] Marin, Gabriel, and John Mellor-Crummey. "Cross-Architecture Performance Predictions for Scientific Applications Using Parameterized Models." SIGMETRICS/Performance '04. 12 – 16 June 2004. New York: Columbia University, 2004. 2 – 13.

[30] Mauer, Hans, et al. Top 500 Supercomputer Sites. 2004. Top 500 Supercomputer Sites. 26 Nov. 2004. <http://www.top500.org/lists/linpack.php>.

[31] Moore, Shirley, et al. "Improving Time to Solution with Automated Performance Analysis." Proceedings of the Second workshop on Productivity and Performance in High-End Computing. 13 Feb. 2005. San Francisco: Palace Hotel, Eleventh International Symposium on High Performance Computer Architecture, 2005. 20 – 26.

[32] Numerical Aerodynamic Simulation Power Benchmarks. Vers. 2.4. Computer software. NASA Ames Research Center, 1990. MPI. Source code.

[33] Nystrom, Nicholas A., John Urbanic and Christina Savinell. "Understanding Productivity Through Non-intrusive Instrumentation and Statistical Learning."

Proceedings of the Second workshop on Productivity and Performance in High-End Computing. 13 Feb. 2005. San Francisco: Palace Hotel, Eleventh International Symposium on High Performance Computer Architecture, 2005. 53 – 61.

[34] Snavely, A, et al. "Performance Modeling of HPC Applications." Proceedings of ParCo 2003. 2 – 5 Sept. 2003. Dresden: Center for High Performance Computing, Dresden University of Technology, 2003.

[35] Snavely, Allan, et al. "A Framework for Performance Modeling and Prediction." Proceedings of SC 2002. 16 – 22 Nov. 2002. Baltimore: Baltimore Convention Center, 2002.

[36] Snavely, Allan, Nicole Wolter, and Laura Carrington. "Modeling Application Performance by Convolving Machine Signatures with Application Profiles." IEEE Workshop on Workload Characterization. 2 Dec. 2001. Austin: ACES Building, University of Texas, 2001.

[37] Squires, Susan, Walter F. Tichy, and Lawrence Votta. "What Do Programmers of Parallel Machines Need? A Survey." Proceedings of the Second Workshop on Productivity and Performance in High-End Computing (PPHEC-05). 12 – 16 Feb. 2005. San Francisco: HPCA05, Palace Hotel, 2005.

[38] Taylor, Valerie E. "Analysis and Modeling of Parallel and Distributed Applications." Seminar. Lawrence Livermore National Laboratory Institute for Scientific Computing Research. Livermore, CA, June 14, 2005.

[39] Tvrdik, Pavel. CS838: Topics in Parallel Computing. "Section \#2: PRAM Models." 23 Jan. 1999. University of Wisconsin – Madison. 10 Dec. 2004. <http://www.cs.wisc.edu/~tvrdik/2/html/Section2.html>.

[40] Varga, András, Y. Ahmet Şekercioğlu, and Gregory K. Egan. "A Practical Efficiency Criterion for the Null Message Algorithm." Oct 2003. European Simulation Symposium 2003. 9 July 2004. <http://ctieware.eng.monash.edu.au/twiki/pub/Simulation/ParallelSimulation/nmaperf.pdf >.

[41] Vetter, Jeffrey, and Chris Chambreau. "mpiP: Lightweight, Scalable MPI Profiling." 25 Jan. 2005. Lawrence Livermore National Laboratories. 19 Jul. 2005. <http://www.llnl.gov/CASC/mpip/>.

[42] Zheng, Gengbin. "Achieving High Performance on Extremely Large Parallel Machines." Diss. U. of Illinois at Urbana-Champaign, 2005. <http://charm.cs.uiuc.edu/papers/GengbinThesis.shtml>.

Appendix B –NPB Spreadsheets

Measured and modeled values with analysis to support graphical data presented in Section V.

CG A

Label	Symbol	Derivation	Unit	Type	1	4	16	64
Collected Values								
Parameters								
Number of CPUs	P	-	CPU	IO	1	4	16	64
Application	-	-	Text	I	CG A	CG A	CG A	CG A
Machine	-	-	Text	I	kc	kc	kc	kc
Run Date	-	-	Date	I	6/16/05	6/16/05	6/16/05	6/16/05
mpiP Collector PID	-	-	#	I	1349	5060	9883	15026
mpiP								
Aggregate Application Time	App_Time	-	s	I	35.5E+0	49.7E+0	65.2E+0	101.0E+0
Aggregate MPI Time	MPI_Time	-	s	I	50.0E-6	3.05E+0	13.3E+0	41.2E+0
Aggregate MPI_WAIT	MPI_Wait	-	s	A	000.0E+0	897.322E-3	4.47E+0	36.813E+0
Number of Messages Sent	M	-	msg	A	1.0E+0	6.724E+3	47.12E+3	269.376E+3
Average Sent Message Size	L	-	B	A	8.0E+0	27.721E+3	11.87E+3	5.539E+3

NAS-PB								
Elapsed Time	-	-	s	I	32.33E+0	11.24E+0	3.62E+0	1.33E+0
Mop/s	-	-	Mop/s	I	46.28E+0	133.1E+0	413.08E+0	1.123E+3
Mop/s/process	-	-	Mop/s	I	46.28E+0	33.28E+0	25.82E+0	17.54E+0
Network Info.								
Bandwidth	BW	(Liner interpolation)	B/s	I	920.0E+3	156.978E+6	105.416E+6	74.338E+6
Latency	Lat	(Liner interpolation)	s	I	8.29E-6	165.253E-6	104.789E-6	69.621E-6
Analysis Views								
Aggregate								
Application Time	App_Time	-	s	I	35.5E+0	49.7E+0	65.2E+0	101.0E+0
MPI Time	MPI_Time	-	s	I	50.0E-6	3.05E+0	13.3E+0	41.2E+0
Non-MPI Time	Non_MPI	App_Time - MPI_Time	s	C	35.5E+0	46.65E+0	51.9E+0	59.8E+0
MPI_WAIT	MPI_Wait	-	s	A	000.0E+0	897.322E-3	4.47E+0	36.813E+0
MPI Active Time	MPI_Active	MPI_Time - MPI_Wait	s	C	50.0E-6	2.153E+0	8.83E+0	4.387E+0

Per CPU								
Application Time	AT*	App_Time / P	s	CIV	35.5E+0	12.425E+0	4.075E+0	1.578E+0
MPI Time	MT*	MPI_Time / P	s	CIV	50.0E-6	762.5E-3	831.25E-3	643.75E-3
Non-MPI Time	-	(App_Time - MPI_Time) / P	s	C	35.5E+0	11.663E+0	3.244E+0	934.375E-3
MPI_WAIT	WT*	MPI_Wait / P	s	C	000.0E+0	224.33E-3	279.363E-3	575.201E-3
MPI Active Time	-	(MPI_Time - MPI_Wait) / P	s	C	50.0E-6	538.17E-3	551.887E-3	68.549E-3
Per Sent Message								
Application Time	-	App_Time / M	s	C	35.5E+0	7.391E-3	1.384E-3	374.941E-6
MPI Time	-	MPI_Time / M	s	C	50.0E-6	453.599E-6	282.258E-6	152.946E-6
Non-MPI Time	MsgCompute	(App_Time - MPI_Time) / M	s	CO	35.5E+0	6.938E-3	1.101E-3	221.995E-6
MPI_WAIT	-	MPI_Wait / M	s	C	000.0E+0	133.451E-6	94.86E-6	136.66E-6
MPI Active Time	-	(MPI_Time - MPI_Wait) / M	s	C	50.0E-6	320.148E-6	187.398E-6	16.286E-6

Per CPU per Sent Message								
Application Time	-	App_Time / (P * M)	s	C	35.5E+0	1.848E-3	86.481E-6	5.858E-6
MPI Time	-	MPI_Time / (P * M)	s	C	50.0E-6	113.4E-6	17.641E-6	2.39E-6
Non-MPI Time	-	(App_Time - MPI_Time) / (P * M)	s	C	35.5E+0	1.734E-3	68.84E-6	3.469E-6
MPI_WAIT	-	MPI_Wait / (P * M)	s	CV	000.0E+0	33.363E-6	5.929E-6	2.135E-6
MPI Active Time	CPUMsgActive	(MPI_Time - MPI_Wait) / (P * M)	s	CO	50.0E-6	80.037E-6	11.712E-6	254.474E-9
Network View								
Per Network Switch								
Switch Delay	D_0	L/BW + Lat	s	CO	16.986E-6	341.845E-6	217.394E-6	144.133E-6
Model View								
Model Inputs								
Customers	N	P	#	B	1.0E+0	4.0E+0	16.0E+0	64.0E+0
Centers	K	P + 2	#	B	3.0E+0	6.0E+0	18.0E+0	66.0E+0
Switch Delay	D_0	L/BW + Lat	s	B	16.986E-6	341.845E-6	217.394E-6	144.133E-6
CPU Service Demand	D_k	(MPI_Time - MPI_Wait) / (P * M) = CPU Message Active	s	B	50.0E-6	80.037E-6	11.712E-6	254.474E-9
Computation Delay	D_{P+1}	(App_Time - MPI_Time) / M = MsgCompute	s	B	35.5E+0	6.938E-3	1.101E-3	221.995E-6

Model Outputs								
System Response Time	R	-	s	R	35.5E+0	7.61E-3	1.53E-3	383.125E-6
Switch Response Time	R_0	-	s	R	16.986E-6	341.845E-6	217.394E-6	144.133E-6
CPU Response Time	R_k	-	s	R	50.0E-6	82.617E-6	13.219E-6	265.58E-9
Computation Response Time	R_{P+1}	-	s	R	35.5E+0	6.938E-3	1.101E-3	221.995E-6
System Throughput	X	-	msg/s	R	28.169E-3	525.603E+0	10.458E+3	167.047E+3
Switch Utilization	U_0	-	#	R	478.478E-9	179.675E-3	2.274E+0	24.077E+0
CPU MPI Utilization	U_k	-	#	R	1.408E-6	42.068E-3	122.487E-3	42.509E-3
Total Computation Utilization	U_{P+1}	-	#	R	999.998E-3	3.647E+0	11.514E+0	37.084E+0

Validation View

Application (Wall Clock) Time								
App Time - Observed (Wall Clock)	AT^*	App_Time / P	s	CI	35.5E+0	12.425E+0	4.075E+0	1.578E+0
App Time - Model	AT	(R * M) / P	s	CR	35.5E+0	12.793E+0	4.506E+0	1.613E+0
Relative Error	E_{AT}	$(AT - AT^*) / AT^*$	%	C	0.0%	3.0%	10.6%	2.2%

MPI Active Time								
MPI Time - Observed	MT^*	MPI_Time / P	s	CI	50.0E-6	762.5E-3	831.25E-3	643.75E-3
MPI Time - Model	MT	$(R_k * M) + (R_0 * M) / P$	s	CR	66.986E-6	1.13E+0	1.263E+0	678.197E-3
Relative Error	E_{MT}	$(MT - MT^*) / MT^*$	%	C	34.0%	48.2%	52.0%	5.4%
MPI Wait Time								
MPI_Wait Time - Estimated	WT^*	MPI_Wait / P	s	C	000.0E+0	224.33E-3	279.363E-3	575.201E-3
MPI_Wait Time - Model	WT	$(R_k - D_k) * M + (R_0 * M) / P$	s	CR	16.986E-6	591.99E-3	711.218E-3	609.647E-3
Relative Error	E_{WT}	$(WT - WT^*) / WT^*$	%	C	#DIV/0!	163.9%	154.6%	6.0%
Throughput								
Throughput - Observed	X^*	M / AT^*	msg/s	C	28.169E-3	541.167E+0	11.563E+3	170.694E+3
Throughput - Model	X	-	msg/s	R	28.169E-3	525.603E+0	10.458E+3	167.047E+3
Relative Error	E_X	$(X - X^*) / X^*$	%	C	0.0%	-2.9%	-9.6%	-2.1%

Graphical View								
Measured Component Time								
MPI Wait	-	MPI_Wait	s	G	000.0E+0	897.322E-3	4.47E+0	36.813E+0
MPI Active	-	MPI_Time - MPI_Wait	s	G	50.0E-6	2.153E+0	8.83E+0	4.387E+0
Computation	-	Non-MPI Time	s	G	35.5E+0	46.65E+0	51.9E+0	59.8E+0
Modeled Component Time								
Switch Delay	-	$R_0 * M$	s	G	16.986E-6	2.299E+0	10.244E+0	38.826E+0
MPI Contention	-	$(R_k - D_k) * M * P$	s	G	-1.263E-12	69.392E-3	1.136E+0	191.458E-3
MPI Active	-	$D_k * M * P$	s	G	50.0E-6	2.153E+0	8.83E+0	4.387E+0
Compute Time	-	MsgCompute * M	s	G	35.5E+0	46.65E+0	51.9E+0	59.8E+0

CG B

Label	Symbol	Derivation	Unit	Type	1	4	16	64
Collected Values								
Parameters								
Number of CPUs	P	-	CPU	IO	1	4	16	64
Application	-	-	Text	I	CG B	CG B	CG B	CG B
Machine	-	-	Text	I	kc	kc	kc	kc
Run Date	-	-	Date	I	6/16/05	6/16/05	6/17/05	6/17/05
mpiP Collector PID	-	-	#	I	16705	15936	19051	21097
mpiP								
Aggregate Application Time	App_Time	-	s	I	2.14E+3	2.84E+3	2.48E+3	2.94E+3
Aggregate MPI Time	MPI_Time	-	s	I	64.0E-6	125.0E+0	371.0E+0	812.0E+0
Aggregate MPI_WAIT	MPI_Wait	-	s	A	000.0E+0	28.185E+0	92.25E+0	249.803E+0
Number of Messages Sent	M	-	msg	A	1.0E+0	31924	223.76E+3	1.279E+6
Average Sent Message Size	L	-	B	A	8.0E+0	148.557E+3	63.587E+3	29.661E+3

NAS-PB								
Elapsed Time	-	-	s	I	2.099E+3	695.48E+0	151.12E+0	44.04E+0
Mop/s	-	-	Mop/s	I	26.07E+0	78.66E+0	362.01E+0	1.242E+3
Mop/s/process	-	-	Mop/s	I	26.07E+0	19.67E+0	22.63E+0	19.41E+0
Network Info.								
Bandwidth	BW	-	B/s	I	920.0E+3	206.038E+6	189.205E+6	160.884E+6
Latency	Lat	-	s	I	8.29E-6	685.107E-6	319.528E-6	173.6E-6
Analysis Views								
Aggregate								
Application Time	App_Time	-	s	I	2.14E+3	2.84E+3	2.48E+3	2.94E+3
MPI Time	MPI_Time	-	s	I	64.0E-6	125.0E+0	371.0E+0	812.0E+0
Non-MPI Time	Non_MPI	App_Time - MPI_Time	s	C	2.14E+3	2.715E+3	2.109E+3	2.128E+3
MPI_WAIT	MPI_Wait	-	s	A	000.0E+0	28.185E+0	92.25E+0	249.803E+0
MPI Active Time	MPI_Active	MPI_Time - MPI_Wait	s	C	64.0E-6	96.815E+0	278.75E+0	562.197E+0

Per CPU								
Application Time	AT*	App_Time / P	s	CIV	2.14E+3	710.0E+0	155.0E+0	45.938E+0
MPI Time	MT*	MPI_Time / P	s	CIV	64.0E-6	31.25E+0	23.188E+0	12.688E+0
Non-MPI Time	-	(App_Time - MPI_Time) / P	s	C	2.14E+3	678.75E+0	131.813E+0	33.25E+0
MPI_WAIT	WT*	MPI_Wait / P	s	C	000.0E+0	7.046E+0	5.766E+0	3.903E+0
MPI Active Time	-	(MPI_Time - MPI_Wait) / P	s	C	64.0E-6	24.204E+0	17.422E+0	8.784E+0
Per Sent Message								
Application Time	-	App_Time / M	s	C	2.14E+3	88.961E-3	11.083E-3	2.298E-3
MPI Time	-	MPI_Time / M	s	C	64.0E-6	3.916E-3	1.658E-3	634.724E-6
Non-MPI Time	MsgCompute	(App_Time - MPI_Time) / M	s	CO	2.14E+3	85.046E-3	9.425E-3	1.663E-3
MPI_WAIT	-	MPI_Wait / M	s	C	000.0E+0	882.89E-6	412.272E-6	195.266E-6
MPI Active Time	-	(MPI_Time - MPI_Wait) / M	s	C	64.0E-6	3.033E-3	1.246E-3	439.458E-6

Per CPU per Sent Message								
Application Time	-	App_Time / (P * M)	s	C	2.14E+3	22.24E-3	692.706E-6	35.908E-6
MPI Time	-	MPI_Time / (P * M)	s	C	64.0E-6	978.887E-6	103.627E-6	9.918E-6
Non-MPI Time	-	(App_Time - MPI_Time) / (P * M)	s	C	2.14E+3	21.261E-3	589.08E-6	25.991E-6
MPI_WAIT	-	MPI_Wait / (P * M)	s	CV	000.0E+0	220.722E-6	25.767E-6	3.051E-6
MPI Active Time	CPUMsgActive	(MPI_Time - MPI_Wait) / (P * M)	s	CO	64.0E-6	758.165E-6	77.86E-6	6.867E-6
Network View								
Per Network Switch								
Switch Delay	D_0	L/BW + Lat	s	CO	16.986E-6	1.406E-3	655.603E-6	357.963E-6
Model View								
Model Inputs								
Customers	N	P	#	B	1.0E+0	4.0E+0	16.0E+0	64.0E+0
Centers	K	P + 2	#	B	3.0E+0	6.0E+0	18.0E+0	66.0E+0
Switch Delay	D_0	L/BW + Lat	s	B	16.986E-6	1.406E-3	655.603E-6	357.963E-6
CPU Service Demand	D_k	(MPI_Time - MPI_Wait) / (P * M) = CPU Message Active	s	B	64.0E-6	758.165E-6	77.86E-6	6.867E-6
Computation Delay	D_{P+1}	(App_Time - MPI_Time) / M = MsgCompute	s	B	2.14E+3	85.046E-3	9.425E-3	1.663E-3

Model Outputs								
System Response Time	R	-	s	R	2.14E+3	89.563E-3	10.399E-3	2.559E-3
Switch Response Time	R_0	-	s	R	16.986E-6	1.406E-3	655.603E-6	357.963E-6
CPU Response Time	R_k	-	s	R	64.0E-6	777.754E-6	79.636E-6	8.405E-6
Computation Response Time	R_{P+1}	-	s	R	2.14E+3	85.046E-3	9.425E-3	1.663E-3
System Throughput	X	-	msg/s	R	467.29E-6	44.661E+0	384.647E+0	25.011E+3
Switch Utilization	U_0	-	#	R	7.937E-9	62.794E-3	252.176E-3	8.953E+0
CPU MPI Utilization	U_k	-	#	R	29.907E-9	33.861E-3	29.949E-3	174.25E-3
Total Computation Utilization	U_{P+1}	-	#	R	1.0E+0	3.798E+0	3.625E+0	41.593E+0
Validation View								
Application (Wall Clock) Time								
App Time - Observed (Wall Clock)	AT^*	App_Time / P	s	CI	2.14E+3	710.0E+0	155.0E+0	45.938E+0
App Time - Model	AT	(R * M) / P	s	CR	2.14E+3	714.802E+0	145.432E+0	51.15E+0
Relative Error	E_{AT}	$(AT - AT^*) / AT^*$	%	C	0.00%	0.68%	-6.17%	11.35%

MPI Active Time								
MPI Time - Observed	MT^*	MPI_Time / P	s	CI	64.0E-6	31.25E+0	23.188E+0	12.688E+0
MPI Time - Model	MT	$(R_k * M) + (R_0 * M) / P$	s	CR	80.986E-6	36.05E+0	26.988E+0	17.908E+0
Relative Error	E_{MT}	$(MT - MT^*) / MT^*$	%	C	26.54%	15.36%	16.39%	41.15%
MPI Wait Time								
MPI_Wait Time - Estimated	WT^*	MPI_Wait / P	s	C	000.0E+0	7.046E+0	5.766E+0	3.903E+0
MPI_Wait Time - Model	WT	$(R_k - D_k) * M + (R_0 * M) / P$	s	CR	16.986E-6	11.847E+0	9.566E+0	9.124E+0
Relative Error	E_{WT}	$(WT - WT^*) / WT^*$	%	C	#DIV/0!	68.12%	65.91%	133.75%
Throughput								
Throughput - Observed	X^*	M / AT^*	msg/s	C	467.29E-6	44.963E+0	1.444E+3	27.849E+3
Throughput - Model	X	-	msg/s	R	467.29E-6	44.661E+0	384.647E+0	25.011E+3
Relative Error	E_X	$(X - X^*) / X^*$	%	C	0.00%	-0.67%	-73.36%	-10.19%

Graphical View								
Measured Component Time								
MPI Wait	-	MPI_Wait	s	G	000.0E+0	28.185E+0	92.25E+0	249.803E+0
MPI Active	-	MPI_Time - MPI_Wait	s	G	64.0E-6	96.815E+0	278.75E+0	562.197E+0
Computation	-	Non-MPI Time	s	G	2.14E+3	2.715E+3	2.109E+3	2.128E+3
Modeled Component Time								
Switch Delay	-	$R_0 * M$	s	G	16.986E-6	44.885E+0	146.698E+0	457.941E+0
MPI Contention	-	$(R_k - D_k) * M * P$	s	G	-161.584E-15	2.501E+0	6.358E+0	125.974E+0
MPI Active	-	$D_k * M * P$	s	G	64.0E-6	96.815E+0	278.75E+0	562.197E+0
Compute Time	-	MsgCompute * M	s	G	2.14E+3	2.715E+3	2.109E+3	2.128E+3

CG C

Label	Symbol	Derivation	Unit	Type	4	16	64
Collected Values							
Parameters							
Number of CPUs	P	-	CPU	IO	4	16	64
Application	-	-	Text	I	CG C	CG C	CG C
Machine	-	-	Text	I	kc	kc	kc
Run Date	-	-	Date	I	6/24/05	6/17/05	6/17/05
mpiP Collector PID	-	-	#	I	1436	2482	8876
mpiP							
Aggregate Application Time	App_Time	-	s	I	9.23E+3	8.63E+3	7.91E+3
Aggregate MPI Time	MPI_Time	-	s	I	121.0E+0	787.0E+0	1.92E+3
Aggregate MPI_WAIT	MPI_Wait	-	s	A	7.67E+0	164.455E+0	595.254E+0
Number of Messages Sent	M	-	msg	A	31.924E+3	223.76E+3	1.279E+6
Average Sent Message Size	L	-	B	A	297.11E+3	127.17E+3	59.318E+3

NAS-PB							
Elapsed Time	-	-	s	I	2.265E+3	527.45E+0	118.43E+0
Mop/s	-	-	Mop/s	I	63.29E+0	271.78E+0	1.21E+3
Mop/s/process	-	-	Mop/s	I	15.82E+0	16.99E+0	18.91E+0
Network Info.							
Bandwidth	BW	-	B/s	I	213.469E+6	204.124E+6	186.148E+6
Latency	Lat	-	s	I	1.325E-3	593.033E-6	301.164E-6
Analysis Views							
Aggregate							
Application Time	App_Time	-	s	I	9.23E+3	8.63E+3	7.91E+3
MPI Time	MPI_Time	-	s	I	121.0E+0	787.0E+0	1.92E+3
Non-MPI Time	Non_MPI	App_Time - MPI_Time	s	C	9.109E+3	7.843E+3	5.99E+3
MPI_WAIT	MPI_Wait	-	s	A	7.67E+0	164.455E+0	595.254E+0
MPI Active Time	MPI_Active	MPI_Time - MPI_Wait	s	C	113.33E+0	622.545E+0	1.325E+3

Per CPU							
Application Time	AT*	App_Time / P	s	CIV	2.308E+3	539.375E+0	123.594E+0
MPI Time	MT*	MPI_Time / P	s	CIV	30.25E+0	49.188E+0	30.0E+0
Non-MPI Time	-	(App_Time - MPI_Time) / P	s	C	2.277E+3	490.188E+0	93.594E+0
MPI_WAIT	WT*	MPI_Wait / P	s	C	1.918E+0	10.278E+0	9.301E+0
MPI Active Time	-	(MPI_Time - MPI_Wait) / P	s	C	28.332E+0	38.909E+0	20.699E+0
Per Sent Message							
Application Time	-	App_Time / M	s	C	289.124E-3	38.568E-3	6.183E-3
MPI Time	-	MPI_Time / M	s	C	3.79E-3	3.517E-3	1.501E-3
Non-MPI Time	MsgCompute	(App_Time - MPI_Time) / M	s	CO	285.334E-3	35.051E-3	4.682E-3
MPI_WAIT	-	MPI_Wait / M	s	C	240.259E-6	734.962E-6	465.298E-6
MPI Active Time	-	(MPI_Time - MPI_Wait) / M	s	C	3.55E-3	2.782E-3	1.036E-3

Per CPU per Sent Message							
Application Time	-	App_Time / (P * M)	s	C	72.281E-3	2.411E-3	96.611E-6
MPI Time	-	MPI_Time / (P * M)	s	C	947.563E-6	219.823E-6	23.45E-6
Non-MPI Time	-	(App_Time - MPI_Time) / (P * M)	s	C	71.333E-3	2.191E-3	73.16E-6
MPI_WAIT	-	MPI_Wait / (P * M)	s	CV	60.065E-6	45.935E-6	7.27E-6
MPI Active Time	CPUMsgActive	(MPI_Time - MPI_Wait) / (P * M)	s	CO	887.498E-6	173.887E-6	16.18E-6
Network View							
Per Network Switch							
Switch Delay	D_0	L/BW + Lat	s	CO	2.717E-3	1.216E-3	619.823E-6
Model View							
Model Inputs							
Customers	N	P	#	B	4.0E+0	16.0E+0	64.0E+0
Centers	K	P + 2	#	B	6.0E+0	18.0E+0	66.0E+0
Switch Delay	D_0	L/BW + Lat	s	B	2.717E-3	1.216E-3	619.823E-6
CPU Service Demand	D_k	(MPI_Time - MPI_Wait) / (P * M) = CPU Message Active	s	B	887.498E-6	173.887E-6	16.18E-6
Computation Delay	D_{P+1}	(App_Time - MPI_Time) / M = MsgCompute	s	B	285.334E-3	35.051E-3	4.682E-3

Model Outputs							
System Response Time	R	-	s	R	291.634E-3	39.246E-3	6.528E-3
Switch Response Time	R_0	-	s	R	2.717E-3	1.216E-3	619.823E-6
CPU Response Time	R_k	-	s	R	895.65E-6	186.209E-6	19.166E-6
Computation Response Time	R_{P+1}	-	s	R	285.334E-3	35.051E-3	4.682E-3
System Throughput	X	-	msg/s	R	13.716E+0	407.681E+0	9.803E+3
Switch Utilization	U_0	-	#	R	37.266E-3	495.74E-3	6.076E+0
CPU MPI Utilization	U_k	-	#	R	12.173E-3	70.89E-3	158.617E-3
Total Computation Utilization	U_{P+1}	-	#	R	3.914E+0	14.29E+0	45.899E+0
Validation View							
Application (Wall Clock) Time							
App Time - Observed (Wall Clock)	AT^*	App_Time / P	s	CI	2.308E+3	539.375E+0	123.594E+0
App Time - Model	AT	(R * M) / P	s	CR	2.328E+3	548.86E+0	130.497E+0
Relative Error	E_{AT}	$(AT - AT^*) / AT^*$	%	C	0.9%	1.8%	5.6%

MPI Active Time							
MPI Time - Observed	MT^*	MPI_Time / P	s	CI	30.25E+0	49.188E+0	30.0E+0
MPI Time - Model	MT	$(R_k * M) + (R_0 * M) / P$	s	CR	50.277E+0	58.672E+0	36.908E+0
Relative Error	E_{MT}	$(MT - MT^*) / MT^*$	%	C	66.2%	19.3%	23.0%
MPI Wait Time							
MPI_Wait Time - Estimated	WT^*	MPI_Wait / P	s	C	1.918E+0	10.278E+0	9.301E+0
MPI_Wait Time - Model	WT	$(R_k - D_k) * M + (R_0 * M) / P$	s	CR	21.945E+0	19.763E+0	16.209E+0
Relative Error	E_{WT}	$(WT - WT^*) / WT^*$	%	C	1044.4%	92.3%	74.3%
Throughput							
Throughput - Observed	X^*	M / AT^*	msg/s	C	13.835E+0	414.851E+0	10.351E+3
Throughput - Model	X	-	msg/s	R	13.716E+0	407.681E+0	9.803E+3
Relative Error	E_X	$(X - X^*) / X^*$	%	C	-0.9%	-1.7%	-5.3%

Graphical View

Measured Component Time							
MPI Wait	-	MPI_Wait	s	G	7.67E+0	164.455E+0	595.254E+0
MPI Active	-	MPI_Time - MPI_Wait	s	G	113.33E+0	622.545E+0	1.325E+3
Computation	-	Non-MPI Time	s	G	9.109E+3	7.843E+3	5.99E+3
Modeled Component Time							
Switch Delay	-	$R_0 * M$	s	G	86.738E+0	272.092E+0	792.937E+0
MPI Contention	-	$(R_k - D_k) * M * P$	s	G	1.041E+0	44.113E+0	244.439E+0
MPI Active	-	$D_k * M * P$	s	G	113.33E+0	622.545E+0	1.325E+3
Compute Time	-	MsgCompute * M	s	G	9.109E+3	7.843E+3	5.99E+3

CG D

Label	Symbol	Derivation	Unit	Type	64
Collected Values					
Parameters					
Number of CPUs	P	-	CPU	IO	64
Application	-	-	Text	I	CG D
Machine	-	-	Text	I	kc
Run Date	-	-	Date	I	8/8/05
mpiP Collector PID	-	-	#	I	2883
mpiP					
Aggregate Application Time	App_Time	-	s	I	336.0E+3
Aggregate MPI Time	MPI_Time	-	s	I	30.2E+3
Aggregate MPI_WAIT	MPI_Wait	-	s	A	6.284E+3
Number of Messages Sent	M	-	msg	A	1.7E+6
Average Sent Message Size	L	-	B	A	593.14E+3

NAS-PB					
Elapsed Time	-	-	s	I	5.144E+3
Mop/s	-	-	Mop/s	I	708.25E+0
Mop/s/process	-	-	Mop/s	I	11.07E+0
Network Info.					
Bandwidth	BW	-	B/s	I	217.381E+6
Latency	Lat	-	s	I	2.6E-3
Analysis Views					
Aggregate					
Application Time	App_Time	-	s	I	336.0E+3
MPI Time	MPI_Time	-	s	I	30.2E+3
Non-MPI Time	Non_MPI	App_Time - MPI_Time	s	C	305.8E+3
MPI_WAIT	MPI_Wait	-	s	A	6.284E+3
MPI Active Time	MPI_Active	MPI_Time - MPI_Wait	s	C	23.916E+3

Per CPU					
Application Time	AT*	App_Time / P	s	CIV	5.25E+3
MPI Time	MT*	MPI_Time / P	s	CIV	471.875E+0
Non-MPI Time	-	(App_Time - MPI_Time) / P	s	C	4.778E+3
MPI_WAIT	WT*	MPI_Wait / P	s	C	98.182E+0
MPI Active Time	-	(MPI_Time - MPI_Wait) / P	s	C	373.693E+0
Per Sent Message					
Application Time	-	App_Time / M	s	C	197.636E-3
MPI Time	-	MPI_Time / M	s	C	17.764E-3
Non-MPI Time	MsgCompute	(App_Time - MPI_Time) / M	s	CO	179.872E-3
MPI_WAIT	-	MPI_Wait / M	s	C	3.696E-3
MPI Active Time	-	(MPI_Time - MPI_Wait) / M	s	C	14.068E-3

Per CPU per Sent Message					
Application Time	-	App_Time / (P * M)	s	C	3.088E-3
MPI Time	-	MPI_Time / (P * M)	s	C	277.558E-6
Non-MPI Time	-	(App_Time - MPI_Time) / (P * M)	s	C	2.811E-3
MPI_WAIT	-	MPI_Wait / (P * M)	s	CV	57.751E-6
MPI Active Time	CPUMsgActive	(MPI_Time - MPI_Wait) / (P * M)	s	CO	219.807E-6
Network View					
Per Network Switch					
Switch Delay	D_0	L/BW + Lat	s	CO	5.328E-3
Model View					
Model Inputs					
Customers	N	P	#	B	64.0E+0
Centers	K	P + 2	#	B	66.0E+0
Switch Delay	D_0	L/BW + Lat	s	B	5.328E-3
CPU Service Demand	D_k	(MPI_Time - MPI_Wait) / (P * M) = CPU Message Active	s	B	219.807E-6
Computation Delay	D_{P+1}	(App_Time - MPI_Time) / M = MsgCompute	s	B	179.872E-3

Model Outputs					
System Response Time	R	-	s	R	200.311E-3
Switch Response Time	R_0	-	s	R	5.328E-3
CPU Response Time	R_k	-	s	R	236.112E-6
Computation Response Time	R_{P+1}	-	s	R	179.872E-3
System Throughput	X	-	msg/s	R	319.503E+0
Switch Utilization	U_0	-	#	R	1.702E+0
CPU MPI Utilization	U_k	-	#	R	70.229E-3
Total Computation Utilization	U_{P+1}	-	#	R	57.47E+0

Validation View

Application (Wall Clock) Time					
App Time - Observed (Wall Clock)	AT^*	App_Time / P	s	CI	5.25E+3
App Time - Model	AT	(R * M) / P	s	CR	5.321E+3
Relative Error	E_{AT}	$(AT - AT^*) / AT^*$	%	C	1.4%

MPI Active Time					
MPI Time - Observed	MT^*	MPI_Time / P	s	CI	471.875E+0
MPI Time - Model	MT	$(R_k * M) + (R_0 * M) / P$	s	CR	542.946E+0
Relative Error	E_{MT}	$(MT - MT^*) / MT^*$	%	C	15.1%
MPI Wait Time					
MPI_Wait Time - Estimated	WT^*	MPI_Wait / P	s	C	98.182E+0
MPI_Wait Time - Model	WT	$(R_k - D_k) * M + (R_0 * M) / P$	s	CR	169.254E+0
Relative Error	E_{WT}	$(WT - WT^*) / WT^*$	%	C	72.4%
Throughput					
Throughput - Observed	X^*	M / AT^*	msg/s	C	323.828E+0
Throughput - Model	X	-	msg/s	R	319.503E+0
Relative Error	E_X	$(X - X^*) / X^*$	%	C	-1.3%

Graphical View					
Measured Component Time					
MPI Wait	-	MPI_Wait	s	G	6.284E+3
MPI Active	-	MPI_Time - MPI_Wait	s	G	23.916E+3
Computation	-	Non-MPI Time	s	G	305.8E+3
Modeled Component Time					
Switch Delay	-	$R_0 * M$	s	G	9.058E+3
MPI Contention	-	$(R_k - D_k) * M * P$	s	G	1.774E+3
MPI Active	-	$D_k * M * P$	s	G	23.916E+3
Compute Time	-	MsgCompute * M	s	G	305.8E+3

Data Type Codes

Types	Description
A	Auxillary calculation, done separately
B	Values for building a model
C	Calculated in this spreadsheet
G	Ancillary calculation for graphical representation
I	Input directly from measurement data
R	Results from model
O	Output for building a model
V	Value for validating a model