Performance Variability of Highly Parallel Architectures

William T.C. Kramer[1] and Clint Ryan[2]

[1] Department of Computing Sciences, University of California at Berkeley and the National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory

[2] Department of Computing Sciences, University of California at Berkeley

Abstract. The design and evaluation of high performance computers has concentrated on increasing computational speed for applications. This performance is often measured on a well configured dedicated system to show the best case. In the real environment, resources are not always dedicated to a single task, and systems run tasks that may influence each other, so run times vary, sometimes to an unreasonably large extent. This paper explores the amount of variation seen across four large distributed memory systems in a systematic manner. It then analyzes the causes for the variations seen and discusses what can be done to decrease the variation without impacting performance.

1 Introduction

The design and evaluation of high performance computers concentrates on increasing computational performance for applications. Performance is often measured on a well configured dedicated system to show the best case. In the real environment, resources are not always dedicated to a single task, and system run multiple tasks that may influence each other, so run times vary, sometimes to an unreasonably large extent. This paper explores the amount of variation seen across four large distributed memory systems in a systematic manner.[1] It then analyzes the causes for the variations seen and discusses what can be done to decrease the variation without impacting performance.

2 Motivation

Application performance for computer systems, including parallel system hardware and architectures, is well studied. Many architectural features are assessed with application benchmarks — be it on real or simulated systems — so the impact of the functions can be evaluated. To a lesser degree, system software is evaluated for performance of both the code itself and the applications that use it.

Despite the studies there is a crucial area of performance that has not received enough attention — specifically how much variation in performance exists in

systems and what contributes to that variation, particularly when software and hardware are viewed as a single system. The variability of performance is a reliability issue as importance as availability and mean time between failures. The user's productivity is impacted at least as much when performance varies by 100% as when system availability is only 50%.

Large scale applications run on multiple CPUs. Many of the most challenging applications, including climate, combustion, material science and life science applications need hundreds to thousands of processors for the most demanding problems. Currently, the only technology that can address these large-scale problems is distributed memory systems consisting of clusters of Shared Memory Processing (SMP) nodes, connected with dedicated interconnect networks. The components of the system that can be adjusted for the application workload, typically fall into three major categories.

- Intensive computation with large memory and fast communication
- Intensive computation with limited memory and less communication
- Intensive computing with large data storage

Architectural features contribute to the performance and variation differently within these broad categories.

3 The Impact of Variable Performance

Large variability of performance has a negative impact on the application user, creating lower user productivity and lower system efficiency. Lower productivity results from several factors. First, the longer a task takes, the longer before the user has a usable result and can go on with analysis. Since some applications have a strict order of processing steps (i.e. in climate studies, year 1 has to be simulated before year 2 can start) longer time steps mean slower progress.

Another loss of productivity is that in most large high performance systems, batch processing is the norm. Jobs (equivalent to the POSIX definition of a UNIX session) are scheduled with a time limits which, if exceeded, cause job termination. With highly variable systems, users are forced to choose between being highly conservative about how much work can be done in a given length of time or risking the loss of work when jobs abort. Both cases cause the user and sometimes the system to be less productive.

Systems become less efficient when high variation exists. Besides the lost work noted above, variation causes less work to flow through the system. Furthermore, most scheduling software relies on user-provided run estimates, or times assigned by default values, to schedule work. When a conservative user over estimates run time, the job scheduler operates on poor information and can cause inefficient scheduling selections on system.

4 Variation Exists in Application Performance

Performance variation is caused by many factors. On multi-user systems with multiple jobs running within a shared memory processor, frequent causes are

memory contention, overloading the system with work, and priorities of other users.

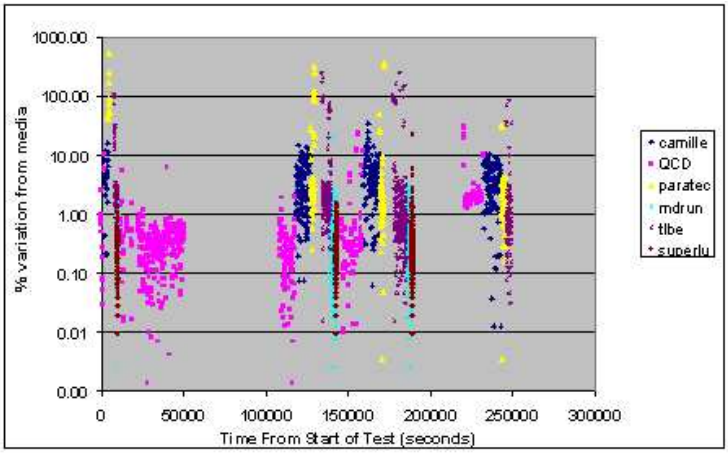

Fig. 1. The variation in performance of six full applications that were part of the NERSC IBM running with 256-way concurrency SP benchmark suite used for system acceptance. The codes were run over a three day period with very little else on the system. The run time variation shows that large-scale parallel systems exhibit significant variation unless carefully designed and configured.

However, on large-scale distributed memory systems, it is rare the compute-intensive parallel applications share SMP nodes. The NERSC IBM SP system, "Seaborg," is a 3,328-processor system with Power 3+ CPUs. It uses an IBM "Colony" switch to connect the SMP nodes. The most efficient manner for applications to use this type of systems is to have nodes dedicated to a single parallel application that uses communication libraries such as MPI to communicate between tasks running in parallel. Thus, many factors normally contributing to variation are not present on these systems, yet, as shown below, application run times can still vary widely.

Figure 1 shows the variation present on the NERSC IBM SP system when it was first installed. Previous experience had shown that a number of software factors could cause variation, including slightly different system software installation on the nodes, system management event timing (daemons running at particular times) and application performance tuning. These issues were all mitigated on the system before the period being discussed. However, configuration problems, bugs, and architectural issues remain that influence variance.

Figure 2 is another example of varying run times for a single code before and after a tuning a new switch configuration. The MPI retransmit interval tells an application how long to wait before retransmission of a message. A typical code shows modest improvements in performance but high variability with a more

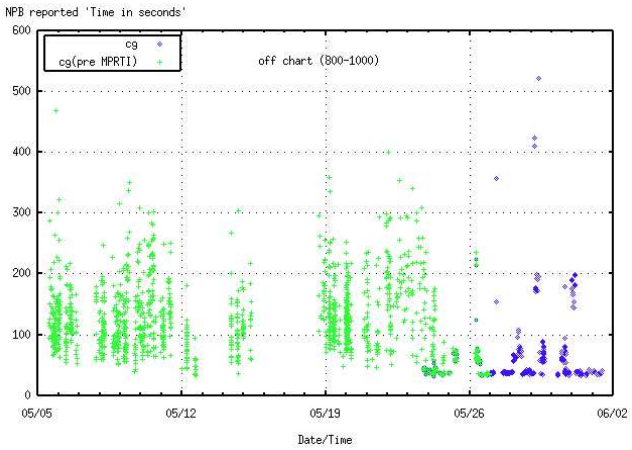

Fig. 2. The variation in performance of the CG benchmark with 256-way concurrency before and after adjustments were made to the MP_RETRANSMIT_INTERVAL interval. The interval controls how long an application waits before retransmitting messages.

aggressive setting. The goal is to recalibrate the timing interval to give the best performance with the least variation. Despite the challenges, some of which are outlined above, it is possible to make such a large system as the NERSC IBM-SP operate in a reliable manner. Table 1 and Figures 3 and 4 show results of running the NAS Parallel Benchmarks on the same system seven months after the period shown in Figures 1 and 2. It shows that in a heavily used (85-95% utilization) system, the benchmarks have consistently low performance variation over multiple runs.

Table 1. Run times (in seconds) reported by the NAS Parallel Benchmarks using 256-way concurrency for the last 50 days of the period covered by Figure 3

Code	Min	Max	Mean	Std Dev	CoV
BT	80.80	84.13	82.42	0.64	0.78%
FT	22.17	23.57	22.42	0.22	0.99%
CG	20.22	24.59	21.39	0.70	3.25%
EP	8.57	8.74	8.61	0.04	0.48%
LU	40.70	43.14	41.75	0.56	1.38%
SP	27.31	28.45	27.73	0.21	0.77%

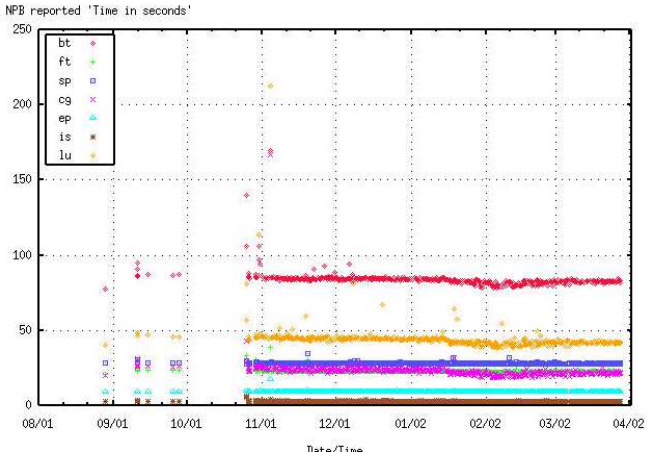

Fig. 3. Seven months of run times for six NPB codes on the same system. The graphs indicate a improvement of performance variation on the system

5 Hypothesis

As noted above, many things can be done to minimize performance variation, including strict system administration and management, nodes dedicated to single application, eliminating bugs, adding more resources and configuration tuning. Nonetheless, fundamental questions remain about how much variation is acceptable, how low variation can be on specific systems, and what most influences variation. This study is the initial attempt to explore the question *Do parallel system architectures/designs influence performance variation in addition to performance itself?*

We believe that architectural and design decisions do contribute to performance variation in significant ways.

The approach of this study is to define experiments to understand factors that determine the relevant key features that influence variability of results. We assembled a set of portable codes that are representative of highly parallel applications and can be used to assess the variability of different system architectures. The codes were run on different architectures and the results were analyzed.

6 The Basic Tests

A number of codes were considered for testing architectures, including SPLASH[2] and Spec[3] benchmarks. Of the benchmark suites available the most effective for this purpose are the NAS Parallel Benchmarks[4] (NPBs), created at NASA Ames Research Center. The benchmarks have been heavily used on a wide range of architectures. They are portable and are known to give comparable results across systems. Since NPBs have been in use for over 10 years (evolving from

Version 1 to Version 2), they are well understood. The codes were sensitive to other activities running on the systems. Finally the benchmarks have been correlated to real scientific workloads at a number of sites.

The NPB benchmarks are implemented primarily in FORTRAN, with some C. They use MPI as the message passing interface. The suite has different problem set sizes, three for parallel code and one for serial execution.

For the sake of simplicity, three NPB benchmarks were chosen for use.

• **LU** — The LU benchmark solves a finite difference discretization of the 3-D compressible Navier-Stokes equations through a block-lower-triangular block-upper- triangular approximate factorization of the original difference scheme. The Computation to Communication ratio is high so the code scales well. LU uses many small messages.

• **FT** — A 3-D FFT PDE with a 3-D array of data is distributed according to z-planes of the array. One or more planes are stored in each processor. The forward 3-D FFT is then performed as multiple 1-D FFTs in each dimension. An array transposition is performed, which amounts to an all-to-all exchange. Thus FT shows big, busty communication patterns amongst all nodes in between periods where all nodes are computing on their own data.

• **EP** — (Embarrassingly Parallel). Each processor independently generates pseudorandom numbers in batches and uses these to compute and tally pairs of normally distributed numbers. No communication is needed until the very end. This test was included to give a baseline for CPU performance.

7 Architectures Evaluated

Four systems with different architectural features were used in the evaluation. The complete features are listed in the Appendix, but a brief summary if provided here.

• **Cray T3E** — The oldest system is the Cray T3E at the NERSC, placed into service in 1997. The "mcurie" system consists of 696 CPUs, each with 256 MB of local memory. Together with interconnect hardware, the processor and local memory form a Processing Element (PE) or node. The PEs are connected by a network arranged in a 3-dimensional Torus with low latency and relatively high bandwidth. The processors are T3E Alpha EV-57 running at 450 MHz, capable of two floating point operations per cycle. The system uses a UNIX like operating system that has a Chorus derived microkernel on the 644 compute nodes and UNICOS/mk on the OS and command nodes. The T3E is the only system with static routing.

• **IBM SP** — The next system is a 3,328 processor IBM-RS/6000-SP at NERSC called "Seaborg." It is composed of a 184 compute nodes containing 16 Power 3+ processors connected to each other with a high bandwidth, switching network known as the "Colony" switch in a Omega topology. A full instance of AIX runs on every node. Each node has two switch adapters. Four nodes have 64 GB, 64 nodes have 32 GB and the rest have 16 GB.

• **Compaq SC** — The Lemieux Compaq SC system at the Pittsburgh Supercomputer Center (PSC) is composed of 750 Compaq Alphaserver ES45 nodes and a separate front end node. Each computational node contains four 1-GHz processors capable of two Flop/s per cycle and runs a full incidence of the Tru64 Unix operating system. A Quadrics Elan3 interconnection network connects the nodes in a Fat Tree topology. Each node is a four-processor SMP, with 4 GB of memory and two switch adapters.
• **Intel** — The final system is LBNL's "Alvarez" commodity cluster of 85 two-way SMP Pentium III nodes connected with Myrinet 2000, another Fat Tree. The CPUs are xSeries 330, running at 866 Mhz with 1 GB SDRAM each. Each node runs Linux RedHat distribution.

Thus, the systems studied show three types of network topology, four operating systems, and four types of processors.

8 Test Results

On each system, a number of runs were executed for each of the three NPB codes. All codes were run using the largest (Class C) problem sets with a 128-way concurrency using 128 MPI tasks. This was chosen because it used at least eight nodes and ran long enough to minimize the effects of start up events. The jobs were run in sets ranging from 10 to 30 runs of each code. Each system allocated dedicated nodes to the tasks. All runs used a one-to-one mapping of CPUs to tasks, which meant that the nodes were fully packed and all CPUs were used. Table 2 summarizes the results for the primary test runs.

Table 2. The basic statistics for the test runs. Including some of the special tests discussed below, over 2,500 test runs were made.

System		**EP**	**LU**	**FT**
Cray T3E	Number of Runs	70	119	118
	Mean Run Time (sec)	35.5	305.2	106.5
	Standard Dev (sec)	2.2	47.8	12.1
	Coefficient of Variance	6.11%	15.58%	11.33%
IBM SP	Number of Runs	424	165	210
	Mean Run Time (sec)	17.4	74.6	41.5
	Standard Dev (sec)	0.09	3.4	2.4
	Coefficient of Variance	0.52%	4.58%	5.70%
Compaq SC	Number of Runs	336	359	371
	Mean Run Time (sec)	5.03	42.8	30.6
	Standard Dev (sec)	0.35	1.9	1.0
	Coefficient of Variance	6.91%	4.53%	3.18%
Intel Cluster	Number of Runs	112	71	119
	Mean Run Time (sec)	17.6	408.7	90.7
	Standard Dev (sec)	0.03	10.7	1.0
	Coefficient of Variance	0.17%	2.62%	1.07%

There was no significant variability due to time of day or which nodes were used by the scheduler to to run the jobs.

9 EP Variation

Two machines, the T3E and PSC Compaq, showed an unexpectedly high variation for EP runs. As its name suggests, EP does very little communication. However, because it has such a short run time, it is possible that individual cases of network congestion caused this variation. If a version of EP that does not use the network still shows significant variation, the individual node must be at fault. To test this, we ran the serial version on the T3E and a four-CPU (a single node) version on the Compaq. The coefficient of variation for the Compaq dropped to less than 1.6%, indicating that something on the node was causing variation. Variation dropped to 1.5% on the T3E, but we wished to determine the effect of migration. We measured only CPU time for the T3E and found less than 0.5% variation. From this we can conclude that the network and, in the case of the T3E, the NPB method of timing, were responsible for most of the variation. Only on the Compaq system do individual nodes contribute to the variation.

10 Changing the Number of Adapters

Two machines in the study, the IBM SP and the Compaq system, have a variable number of network ports (adapters) on each node. We expected the use of more adapters would mean run time and variation for the LU and FT benchmarks, but not for the EP benchmark. A set of test runs was made of all three benchmarks using both one and two adapters.

Changing from one to two adapters had a statistically significant effect on mean run time only for the EP and FT program runs on the SP machine ($p < .01$ in both cases). Using an F-test to compare changes in the variation, we found using two adapters increases variation for the FT benchmark on both systems ($p < .05$ in both cases). Variation for the LU benchmark decreased with two adapters on the Compaq ($p < .01$) and increased with two on the SP ($p < .01$). As expected, changing the adapter had no significant effect on the EP benchmarks.

These results agree with many, but not all, of our hypotheses. Increasing the number of adapters did not have much effect on mean run time, probably because the codes do not send enough data to benefit from an increase in bandwidth. We cannot explain why EP shows an increase in performance on the SP with two adapters. In addition, we do not yet know why LU performed more consistently on the Compaq with two adapters. These results do suggest two things. First, different factors influence variation; one cannot simply say that changing one particular aspect of a network will decrease variation for all programs that use the network. Second, when optimizing a program for a particular system, a programmer should consider things that do not necessarily increase megaflop

rate or decrease memory usage. Analysis of Variance (ANOVA) suggest that changing adapters had the following effects on run time: a) no effect on PSC, b) no effect on SP for the LU code and c) an effect on the SP both the EP and SP codes.

11 High Variation on the Cray T3E

With the exception of the T3E, all of the machines studied had distributions such as the one shown in Figure 5. In essence, the distributions were tight bell curves with long right tails. Almost every job experienced some normally distributed slowdown, while a few suffered significantly more. A typical distribution for the T3E is shown in Figure 6. The majority of jobs experienced only a very small slowdown, while a significant portion suffered a far larger slowdown (40% or more in some cases). To test for this, we examined the system logs for some of the runs and measured only system time. This caused the histograms to collapse into ones similar to Figure 6.

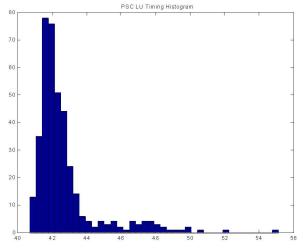

Fig. 4. Histogram of LU times from the Compaq SC system. It shows Gaussian distribution with a long fat tail.

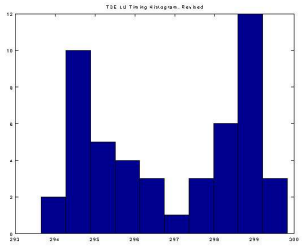

Fig. 5. LU runs from the T3E using the NPB report times. It shows a much tighter distribution.

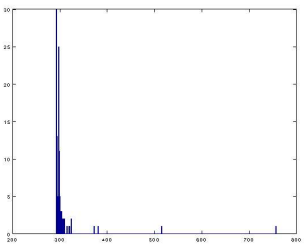

Fig. 6. T3E distribution based on accounting data, not the run time as reported by LU. This eliminates the impact of migration and shows a much more tightly packed distribution.

We were surprised at the variation indicated for the Cray T3E, which seemed unusually high, and investigated further. In order to make efficient use of the network, the T3E assigns logical node numbers to physical PEs at boot time.[5] Physical node numbers are based on how the node physically connects in the interconnect network. Logical numbers are assigned deterministically to minimize routing. The switch does direction order routing and special routing, but adaptive routing was never implemented, so the T3E only routes data through a predefined path using virtual channels.

Jobs are scheduled on logically contiguous nodes. This means that large contiguous blocks of PEs gradually become fragmented, making it increasingly difficult to run jobs requiring large numbers of CPUs. The T3E addresses this problem by periodically scanning all the PEs and identifying ones that have no work assigned. In a manner similar to memory shuffling, the system "migrates" jobs to pack all the running PEs together. This creates larger sets of contiguous PEs for new jobs to start. Jobs are assigned to PEs using a number of parameters, including an alignment measure that indicates how the starting point and/or ending point of the application aligns to power of two logical PE node number.[6]

In order to efficiently schedule new jobs, the T3E system software called the Global Resource Manager (GRM) scans all the nodes to look for opportunities to migrate. The frequency of scans is site selectable; on the T3E under study occur at five second intervals.

When jobs are migrated, system accounting is adjusted to compensate for the time the job is moving and not processing. However, the real time clock continues, which is what is used to report the NPB run times. System accounting logs have been correlated with the output of the NPB tests. Table 3 shows the difference between the real time values and the system accounting time for the job.

The table shows that when adjusted for time spent migrating, the variation of the T3E improves considerably. The situation caused by the T3E having to migrate to maintain a mapping of PEs to the location in the switch fabric has interesting trade offs. Much of the impact of variation discussed earlier is mitigated, since jobs will not abort due to exceed run times. Yet there are

Table 3. Comparison of T3E coefficients of variance using actual NPB Run Time reports and system accounting data. The NPB run time reports calculate the "wall clock" time for the test — and do not adjust for time lost due to migration or checkpoints.

Cray T3E	EP	LU	FT
NPB reported run times using wall clock	6.11%	15.58%	11.33%
System accounting reported run times, not including time spent during job migration	0.8%	0.6%	0.93%

consequences, such as users waiting longer for results. Not migrating also has consequences, since certain work will not progress through the system as rapidly and system productivity will decrease.

12 Detecting and Reacting to Variation

Off-line detection and reaction to variation is possible and is done. Since minimal variation is not typically designed into architectures, most remediation is via system management and software. Making nodes strictly homogenous in hardware is key—but in differing amount so memory is not an issue unless the application tries to exploit virtual memory. Making nodes strictly homogeneous with respect to system software and timing of daemon task runs can have great impact. Indeed some parallel applications take less time if they do not use all the CPUs in a node to avoid the intrusive impact of system tasks that run periodically. Thus a coordinated, system wide timing "heartbeat" to coordinate the execution of system housekeeping tasks on all nodes may be beneficial.

Architecturally, variation may be impacted most by how the interconnect functions. Thus a clear evaluation of the likely variability of messages for different loaded conditions and patterns would seem called for as part of switch design. The effort to drive variation down to single digits after the fact is large and complicated.

The work discussed at NERSC required a team of 12 experts working together for six months, having skills in such areas as switch software and hardware, operating systems, MPI, compilers, mathematical libraries, applications and system administration. The improvements also involved major modifications to the switch micro code, lowest level software drivers and global file systems.

Detecting variation in real time so an application can respond is difficult. Dynamically detecting and responding in the proper manner is even more difficult. Some codes, such as the Gordon Bell Prize-winning LSM, are internally instrumented to report the performance of internal steps — such as reporting the overall performance or length of time taken for a time step of simulation. From

there, it is feasible to consider monitoring the periods and identifying whether the past period is within an appropriate range.

Another way to instrument codes is with tools that acquire information from the hardware. While normally used for debugging and after the fact analysis of performance, such tools conceivably could be used to assess variation. Past systems such as the Cray YMP and C-90 could monitor and report hardware performance with virtually no overhead. In order to assess whether today's tools do the same, the benchmarks were built with the IBM performance tools[7], which monitor and report performance. Seven runs of the three instrumented benchmarks were made on the IBM SP and compared to the runs made without the monitoring software. The codes performed up to 250% slower when instrumented. Performance variation for the instrumented codes also was dramatically higher, with increases between 250% and 400%. This is in part because retrieving information counters in network adapters stops all traffic for a period of time. Unless more efficient methods are developed to decrease the overhead of performance monitoring, it is unlikely applications will be able to use these tools directly.

Even when variation is detectable, it is not clear what the proper action to take would be without a better model of what is going on. For example, it is unclear whether adding more CPUs to an application improves variation, because it changes both network traffic patterns and forces the application to scale more. Likewise, it may be decreasing CPUs would improve the variation and possibly performance. Several teams using large shared codes specifically run special tests on target systems to assess performance tradeoffs. However, it is uncommon for people to also test for tradeoffs in variation. Another concern is the fact that an application spending time monitoring and deciding what to do will have a longer run time and possibly a variation in performance.

13 Further Work

Much more needs to be done to fully understand the issues causing performance variation. The study demonstrates the methods used — running codes across architectures — is a valid and meaningful test. Clearly, adding more systems to the study so there is more overlap of architectural features. This will narrow down features that influence performance variation. One approach is to run the test on systems that have either a different number of processors per node or the same CPUs with a different switch. Another area to study is performance variation differences between shared memory systems and distributed memory system. Being able to check variation across system software versions and upgrades is critical. A different thrust is to develop specialized tests to perform finer grain studies. Such a test would include creating artificial benchmarks containing synchronizing message passing calls that are timed. Measuring which messages reach the barrier first and the distribution of message arrival times at a barrier could lead to further insights.

14 Conclusions

Performance variation does exist on large distributed memory systems and can have a very significant impact on the useability and effectiveness of the system. While difficult, it is possible to constrain but not eliminate wide variation with good system management, tuning and design.

Performance variation still remains a problem and comes from complex tradeoffs of design and implementation. The T3E's need to migrate jobs in order to have contiguous nodes assigned to jobs due to switch routing increases variation. Yet, without migration, system scheduling would be much less efficient. If the architecture supported adaptive routing this conflict might be mitigated.

Architectural features do influence performance variation. The study of one or two adaptors shows architectural features impact performance and variation unexpected ways.

Some of the responsibility to control performance variation belongs in the domain of system managers. They have to assess how configuration and tuning changes will impact performance variation as well as absolute performance.

Responsibility for understanding variation also rests with the application user. The adapter study shows selecting the use of architectural features may not change performance but may positively or negatively influence the variation of performance.

Finally, the study of performance variation is important and can have impact in design of systems and applications. It is worth more resources to continue this understanding.

15 Acknowledgements

Thanks are given to the Pittsburgh Supercomputer Center and the National Energy Research Scientific Computing Center. In particular, David Skinner, Tina Butler, Nicholas Cardo, Adrian Wong Thomas Davis and Jonathon Carter contributed ideas to this paper. Special thanks are also owed to Prof John Kubitowicz for his suggestions and guidance.

References

1. *Strategic Proposal for the National Energy Research Scientific Computing Center FY2002–FY2006*, LBNL Technical Report No. 5465, December 21, 2001. `http://www.nersc.gov/aboutnersc/ pubs/Strategic_Proposal_final.pdf`
2. Woo, Stephan, et al., The SPLASH-2 Programs: Characterization and Methological Considerations, Proc ISCA, June 1995, Santa Margherita Ligure, Italy, pp 24-36.
3. Henning, John l., SPEC CPU 2000: Measuring CPU performance in the New Millenium, IEEE Computer, July 2000, pp 28-35.
4. David H. Bailey, et al., The NAS Parallel Benchmarks, Intl. Journal of Supercomputer Applications, vol. 5, no. 3 (Fall 1991), pg. 66-73
   `http://www.nas.nasa.gov/Research/Reports/Techreports/1996/nas-96-010-abstract.html`

5. Cray T3E System Support Skills, Cray Research Technical documentation, Number R-T3ESSS, Revision R/H, May 1997
6. UNICOS/mk Resource Administration, SG-2602, version 2.0.2
7. Parallel System Support Programs for AIX Administration Guide, Version 3 Release 1.1, SA22-7348-01, 22 Jul 1999
8. IBM Parallel Environment for AIX Dynamic Probe Class Library Programming Guide, Version 3 Release 1, Document Number SA22-7420-00

Table 4. Appendix: Features of Evaluated Architectures

Category	Cray T3E NERSC "mcurie"	IBM SP NERSC "Seaborg"	Compac SC PSC "lemieux"	IBM Netfinity LBNL "alvarez"
CPUs	Alpha EV57 based	Power 3+	Alpha EV 68	Intel Pentium III
Clock (MHz)	450	375	1,000	866
Mflop/s per CPU	900	1,500	2,000	866
CPUs per node	1	16	4	2
Memory/CPU(MB)	256	1,000 to 4,000 most runs on 1,000	1,000	512
Caches	L1–8KB L2–96KB	L1–Data 64KB L1–Inst 32KB	L1–Data 64KB	L1–Data 16KB L2–Inst 16KB
Memory Bandwidth (GB/s)	.96 per CPU	1.6 per CPU Switched Base	8 per node 8 per node (2 per CPU) Switch Based	.532 per CPU Shared Bus
Switch Technology	Custom	IBM "Colony"	Quadrics	Myrinet 2000
Switch Topology	3D Torus	Omega Network	Fat Tree	Fat Tree
Adapters per node	1	2 (2 is default)	2 (1 is default)	1
Interconnect Bandwidth per adapter (MB/s)	300	1,000	280	240
Latency (MPI) (μ sec)	4.26	18.3	~ 5	11.8
Ping-Pong Test MPI to MPI 1 task per node (MB/s)	303	365		150
Number of CPUs	696	3,328	3,000	170
Number of CPUs per node	1	16	4	2
Compilers	Cray	IBM	Compaq	PGH
O/S	Chorus μ kernel	AIX with SP software	Tru 64 Unix	RH Linux
Load	Heavy $> 90\%$	Heavy $> 90\%$	Heavy 75–85%	Very Light usually only user
Peak Aggregate Performance (Tflop/s)	.63	5.0	6.0	.15
Latest LINPACK performance results (Tflop/s)	.48	3.05	4.06	.09
Memory Ratio (B/Flop/s)	.28	.67 (1.3 if a 32 GB node was used)	.5	.6
Communication Ratio (based on default number of adapters) (B/F)	.333	.083	.0425	.138
Aggregate Main Memory Bandwidth for 128 CPUs (GB/s)	122	205	256	68