Evaluating System Effectiveness in High Performance Computing Systems

Adrian T. Wong, Leonid Oliker, William T. C. Kramer,
Teresa L. Kaltz and David H. Bailey

LBNL Technical Report #44542
DRAFT Date: Nov. 11, 1999

Abstract

High performance scientific computer systems traditionally have been compared using individual job performance metrics. However, such metrics tend to ignore high-level system issues, such as how effectively a system can schedule and manage a varied workload, how rapidly the system can launch jobs, and how quickly it can recover from a scheduled or unscheduled system outage. Yet high-level issues such as these are now among the most important factors in the overall effective performance of a system. This article discusses these issues in some detail and then describes a new performance test, called the "Effective System Performance (ESP)" benchmark, which we are developing to measure system-level performance.

Affiliation of all authors: Lawrence Berkeley National Laboratory, NERSC Division, University of California. Send correspondence to David H. Bailey, Lawrence Berkeley Laboratory, Mail Stop 50B-2239, Berkeley, CA 94720, email: dhbailey@lbl.gov. This work was supported by the Office of Computational and Technology Research, Division of Mathematical, Information and Computational Sciences of the U.S. Department of Energy, under contract DE-AC03-76SF00098.

Introduction

Ever since their creation in the mid-1950s, modern scientific computer systems have been compared by measuring their performance on various scientific benchmark codes. Some examples of these benchmarks include the Linpack scalable benchmark [2] and the NAS Parallel Benchmarks [1]. Many large high-performance computing centers have devised additional benchmarks, which are intended to reflect the particular application mix at their sites. System managers rely heavily on these performance figures in selecting new systems and in determining when currently installed systems should be upgraded.

The theoretical peak performance of a system (the product of floating-point operations per clock period, the clock rate, and the number of CPUs) provides only a rough indication as to what performance can be obtained on real-world scientific application programs. The accuracy of this estimate varies widely with computational algorithms, system architectures, software environments and vendors. This figure is cited so frequently only because it is very simple to determine. A more meaningful metric of computational performance is “sustained” performance -- namely, the measured performance of a system on one or more benchmarks that better reflect the real workload. The ratio of these two is often termed the “efficiency” or “percent of peak”.

The percent of peak achieved by various computational applications varies widely, both between systems and even on a single system. For example, the SGI/Cray T3E system at the National Energy Research Scientific Computing Center (NERSC) has a theoretical peak performance rate of 580 Gflop/s (644 CPUs times 900 Mflop/s per CPU = 580 Gflop/s). Taking the average of the NAS Parallel Benchmarks (NPB) performance rates on the T3E, omitting the EP and IS benchmarks and linearly scaling from 256 to 644 CPUs, gives only 29.6 Gflop/s, which is a mere 5.1% of peak. Using another suite of seven benchmarks, which are reflective of codes run on the NERSC computers, the T3E achieved an average of 67 Gflop/s, or about 11.6% of peak. The LSMS code, which was used by Andrew Canning and his colleagues in winning the 1998 Gordon Bell prize competition[5], runs at 256 Gflop/s on the NERSC T3E, or about 44.1% of peak.

A common weakness all these performance benchmarks, whether they give good results or poor, is that they do not measure system-level aspects of performance. A computer system that can achieve an impressive performance rate on an individual benchmark, or even on a suite of benchmarks, is nonetheless of only limited value if large amounts of CPU time are lost in day-to-day operation due to system-level inefficiencies.

For example, there is considerable variation between various systems in the effectiveness of their job scheduling software. In many centers, large grand-challenge jobs are granted increased priority at night. Supporting this type of operation requires a system scheduler that can cleverly assign jobs to “back-fill” the nodes that otherwise would have to be emptied to permit the large job to run. Systems that have such a feature can deliver significantly more resources (compute cycles, I/O, communication) to application scientists and therefore have greater scientific impact. Job launch time is another aspect of system management that varies considerably from system to system. Launch time is of minor significance when it is a second or two, but it looms a major issue if many minutes pass between the time the system initiates the job, but before the job actually commences useful computation.

Both users and system managers would prefer a computer system whose hardware and software never crash. But state-of-the-art systems are unavoidably less stable than more mature systems. And even very stable systems must periodically be taken out of service for scheduled maintenance and system upgrades. In addition, unforeseen environmental problems, such as power interruptions and outages, do happen in real-world operational settings, and these realities must be taken into account when assessing the real throughput performance of a system.

When a system outage does occur, for any reason, one key question is how long it takes for the system to return to full operational status. This can make the difference between an outage being a minor nuisance or a major catastrophe. Along this line, some systems may lose one or a few individual nodes without the entire system being taken out of service. A system that can quickly recognize this situation and initiate the necessary restart procedure, without interfering with jobs running on the rest of the system, has a significant advantage.

Clearly some advanced system software can greatly improve overall system performance and utilization. One key item here is a system-initiated checkpoint-restart facility. This facility can be used, for example, to save the states of a set of smaller jobs, so that a full-system job can execute with minimal interruption and loss of resource. This facility is also very helpful when taking the system out of service for scheduled maintenance.

As a system matures (speaking in terms of both hardware and software), overall system utilization and performance increases. This is due to several factors:

1. Additional system management tools are made available by the vendor.
2. Existing system management tools become more effective and robust.
3. The user workload stabilizes.
4. Users learn how to adjust their jobs to best utilize the system's batch system.
5. System managers learn how to best schedule the user workload and to best use the available system management tools.
6. Compilers, I/O and other system software facilities improve.

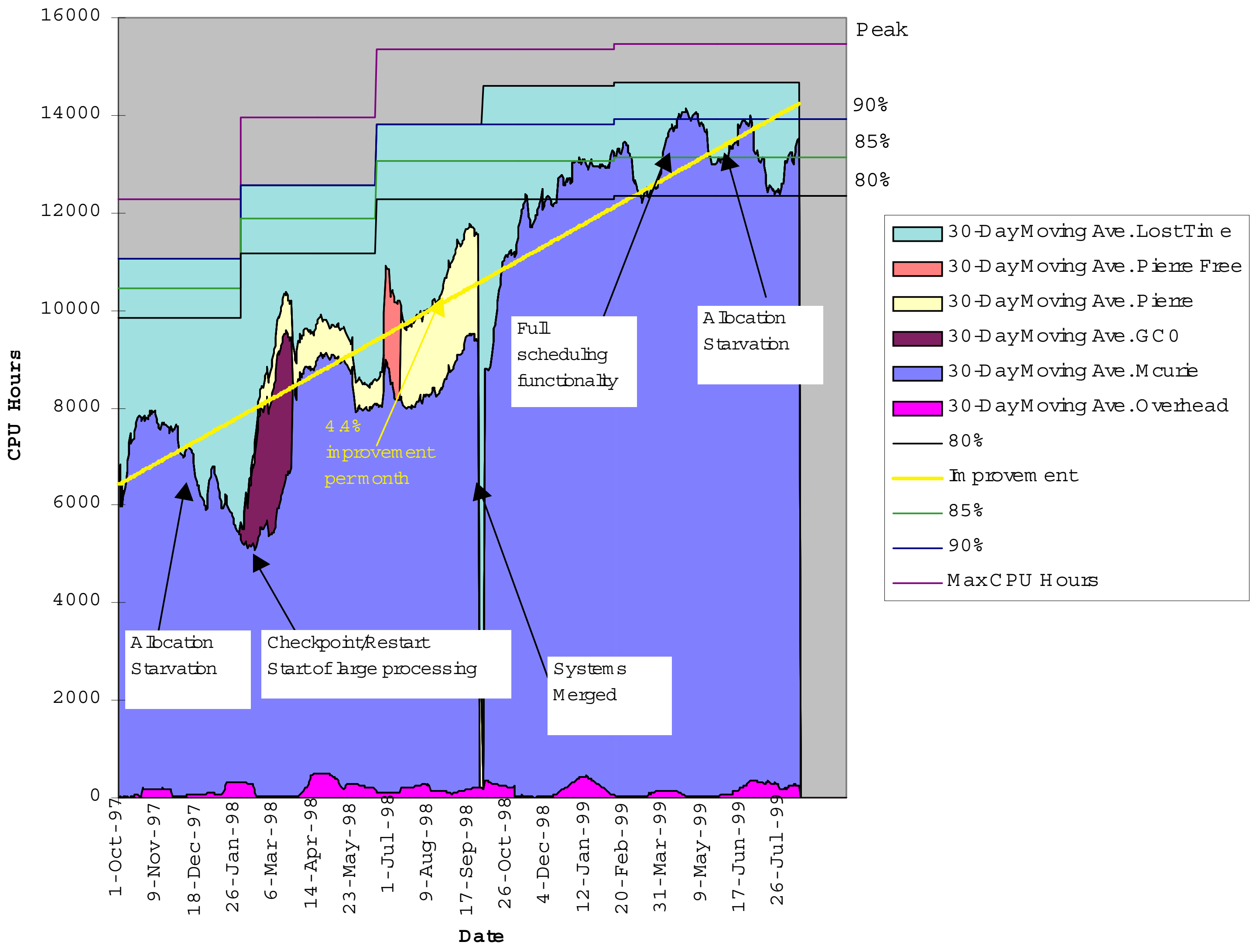
Achieving high levels of system utilization is definitely more challenging on highly parallel, distributed memory computer systems than on conventional shared-memory systems. One reason for this is that conventional systems have traditionally been operated mostly as servers for single-processor jobs. Further, it is very easy for multiple jobs to share the available memory and CPUs on such systems. Highly parallel systems, by comparison, present greater challenges for system management, and successfully addressing these challenges will be a key to the success of these systems in the coming years [5]. Along this line, it is widely believed that traditional shared memory supercomputers can be operated at 90% or higher utilization, but that highly parallel, distributed memory systems cannot possibly achieve such high utilization rates, especially if they focus on grand-challenge, state-of-the-art research jobs. Recently, however, over 90% utilization was achieved on the NERSC T3E for a significant period of time, even though the workload generally emphasizes large, grand-challenge jobs (see Figure 1). This indicates that a high level of system utilization is possible on distributed memory parallel systems -- it is only a matter of adequate software facilities, system maturation and effective management [4].

The Economic Value of Increased System Utilization

Increasing the level of system utilization can produce very real savings. For example, running a 725 CPU system with 80% utilization produces the same total computational capability as running a smaller 644 system at 90% utilization. This is an 81 CPU difference. If we take the figure $25,000 for the unit cost of a CPU on the T3E, purchased in quantity, then a 10% utilization improvement approximately translates into a $2,000,000 cost savings (these figures are current as of November 1999).

The experience of the NERSC computer center underscores this reckoning. Over the 18 months indicated in Figure 1, the utilization of the NERSC T3E increased from roughly 55% to roughly 90%. This can be valued at $10.25 million, which is more than one third the system cost, or about 80% of one Moore's Law generation. In other words, the savings obtained by increases in system utilization has been almost the same as Moore's Law improvements in performance per dollar (up to a point of diminishing marginal returns).

Figure 1: NERSC T3E Utilization



Effective System Performance

We mentioned above the distinction between "theoretical peak" performance and "measured" or "sustained" performance for a user code or benchmark suite. By analogy, we can distinguish the "theoretical maximum" system capacity from "measured" system capacity. Here "theoretical maximum" system capacity is the product of the total number of processors times the total wall clock time that a system is up and available for service (in other words, excluding periods when the system is down). This is analogous to "theoretical peak" performance in that it is a "not-to-be-exceeded" theoretical maximum level. Then by analogy with the above we can define the "measured" or "sustained" system capacity as the sum of wall clock seconds that a node is executing a real job, summed over all nodes. The ratio of these two can be taken as a "system efficiency ratio", which, when multiplied by the sustained performance figure, gives a statistic that we term "effective system performance".

The ESP Test

In order to measure effective system performance, we have developed the Effective System Performance (ESP) test. Some of our objectives in formulating this test include:

1. To determine how well existing systems at our site supports our particular scientific workload.

2. To assess systems before they are purchased.

3. To estimate the approximate performance of a system before it is built, based on design characteristics.

4. To provide quantitative information in making decisions regarding possible enhancements of system hardware or software.

5. To compare different systems in their effectiveness on a workload taken from a single discipline.

6. To compare system-level performance on workloads derived from different disciplines.

The ESP test is not intended to study solutions for the challenges of handling legacy hardware and software.

Definition of the NERSC ESP Test

The ESP test employs of a suite of codes that will be termed here the "Mix". The standard NERSC suite is a set of 82 individual jobs, as shown in Table 1. Two of these are "full configuration" (FC) jobs, namely calculations that use the entire system. The figure in the "Individual T3E time" column is the time required to run one instance of this job on dedicated nodes of the NERSC T3E system. The figure in the "percentage" column is the total run time for the benchmark (namely, the number of instances multiplied by individual T3E run time), as a percentage of the total for all jobs. Note in Table 1 that most, but not all, of the CPUs required for jobs in the Mix suite are powers of two. This is similar to the mix of jobs currently running on the NERSC T3E. In general, the Mix suite reflects the current distribution of jobs on the NERSC T3E, except that the run times have been compressed by a factor of three in order to reduce the total amount of time required for the ESP test, and the discipline mix is not quite the same.

Except for the two FC jobs, these jobs are submitted to the system's job scheduling software in an order given by a particular pseudo-random number generator. At time zero, jobs are submitted in this manner until the total number of CPUs requested by the jobs in the queue exceeds twice the number of CPUs available in the system. Then ten minutes after the start of the test, another batch of pseudo-randomly selected jobs is submitted, until the total number of CPUs requested by the new jobs in the queue exceeds the number of CPUs available. Twenty minutes after the start of the test, the remainder of the jobs in the suite (except for the FC jobs) are submitted. The CPU requirements and expected run times, as shown in Table 1 for the T3E, are to be provided to the system scheduler for all jobs in the test suite.

Twenty-four minutes into the test, the first of the two FC jobs is submitted. This job is to be run immediately upon submission. In particular, it shall run before any other job in the queue that has not already stated executing. This may be achieved by one of the following methods:

1. Disabling the input job queue, waiting until all running jobs have completed, and then running the FC code.
2. Terminating the running jobs, allowing the FC job to run, and then restarting the terminated jobs later.
3. Employing some form of concurrent, priority scheduling system that permits the FC job to be run on nodes currently being used by other jobs.
4. Saving the currently running jobs, using a system-initiated checkpoint-restart facility, allowing the FC job to be run, and then resuming the other jobs later.

Immediately after the FC job is completed, the system shall be entirely shut down and then rebooted. If the elapsed time required for a system shutdown and a restart is known from a previous consistent test run, and if no significant changes have been made to the system in the meantime, the shutdown step need not be actually performed -- this elapsed time is simply added to the total run time for the test. After the system has been rebooted, the Mix suite shall be restarted.

75% into the test, based on an estimated run time, a second FC job shall be submitted. The second FC job must complete no later than 90% of the way through the test. In this case, a back-fill scheme may be utilized in this phase of the test to minimize “white space”.

All codes must run to completion, with verified correct results. The test profile is shown in Figure 5. The run time T is then defined to be the average of the wall clock time when the system first had available resources (ie, available CPU) but was unable to schedule additional jobs because the input queue was empty, and the wall clock time when the system completed all jobs. This average value is specified here as compensation for the fact that part of the “white space” near the end the test is an artifact of the test design, which has a limited set of input jobs rather than a continual stream as in an operational setting.

Code Name	Discipline	Number of of CPUs	Instances	Individual T3E time	Percent of total
gfft	Large FFT	512	2	30.5	0.42
md	Chemistry	8	4	1208.0	0.52
md		24	3	602.7	0.58
nqclarge	Chemistry	8	2	8788.0	1.89
nqclarge		16	5	5879.6	6.32
paratec	Material science	256	1	746.0	2.57
qcdsmall	Nuclear physics	128	1	1155.0	1.99
qcdsmall		256	1	591.0	2.03
scf	Chemistry	32	7	3461.1	10.42
scf		64	10	1751.9	15.08
scfdirect	Chemistry	64	7	5768.6	34.75
scfdirect		81	2	4578.0	9.97
superlu	Linear algebra	8	15	288.3	0.47
tlbebig	Fusion	16	2	2684.5	1.16
tlbebig		32	6	1358.3	3.51
tlbebig		49	5	912.0	3.00
tlbebig		64	8	685.8	4.72
tlbebig		128	1	350.0	0.60

Table 1: The Mix Suite

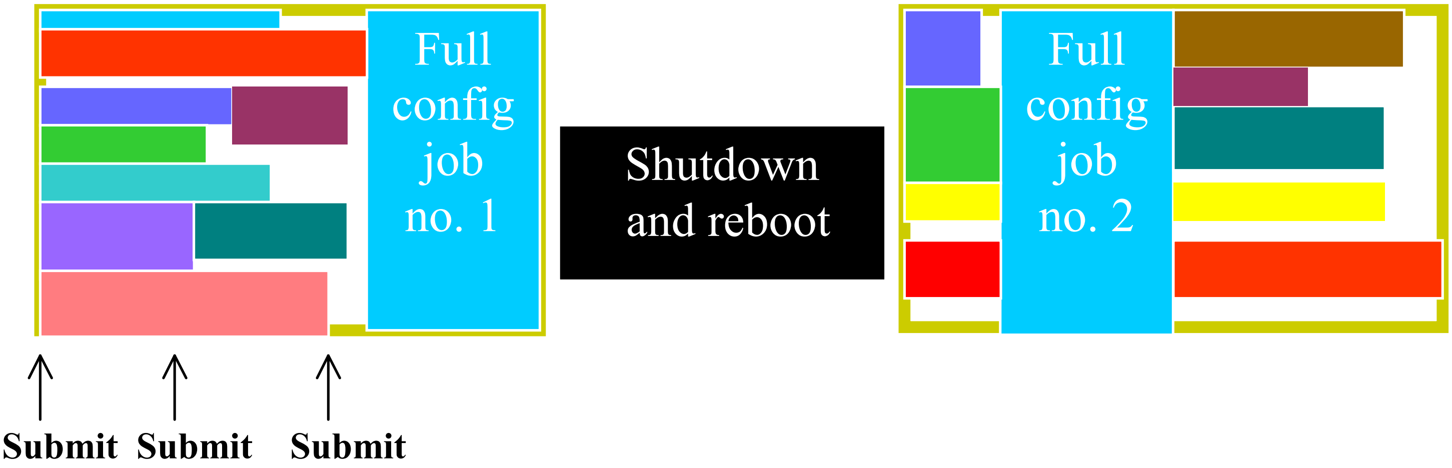


Figure 5: ESP Test Timeline

The system effectiveness ratio E is then computed as

$$E = \frac{p_1 t_1 + p_2 t_2 + ... + p_n t_n}{P \ (S + T)}$$

where

p_i = Number of processors utilized by job i.

t_i = Wall clock run time in seconds required by job i on a dedicated system.

n = Number of individual jobs run during the test.

P = Total number of processors in the system.

S = Time required for shutdown and reboot.

T = Averaged wall clock run time (not counting shutdown and reboot) -- see text.

Note that if the job scheduling system were perfect, if shutdown and reboot times were zero, and if there is no system contention among concurrently running jobs, E would be unity. On real-world systems, E will be less than unity. When multiplied by a sustained performance figure obtained, for instance, by using a standard multiprocessor benchmark, one obtains an "effective system performance" statistic that we believe will more accurately characterize real-world system-level performance.

Variations of the ESP Test

For simplicity, the definition of the ESP as given above is targeted to the NERSC T3E. But clearly the test can easily be modified to make it more appropriate for other systems. For example, on systems with more than 512 CPUs, it would make sense to modify the full configuration jobs to run on more than 512 CPUs, and it would also make sense to increase overall workload accordingly. Also, the distribution of CPU requirements can be changed by altering the numbers of instances of various jobs from those figures given in Table 1.

Other centers may wish to substitute jobs emphasizing different types of scientific applications for those jobs in the standard NERSC Mix suite. Even within a single center, one may study the extent to which jobs from different scientific disciplines affect the overall system-level performance, simply by making appropriate substitutions in the Mix suite.

It should be noted that the baseline run times t_i in the ESP formula are to be measured on the target system in a dedicated or nearly dedicated environment. In this way, the overall ESP system efficiency ratio implicitly includes a measurement of the extent to which jobs running together in a multi-user environment yield lower performance, due to contention effects.

As mentioned above, the CPU requirements and expected run times are to be provided to the system scheduler for all jobs in the test suite. One variation here is for the submission script to randomly select 25% of these jobs, using the pseudo-random number generator mentioned above, and terminate them in less than the scheduled time. This is intended to mimic a real-world operational situation, where significant numbers of jobs terminate early, either due to inaccurate estimates of run time, or to errors in the code.

ESP on the NERSC T3E

The ESP test was run as described above on the NERSC T3E system on October 17, 1999, except that the system was not actually shutdown and rebooted -- an estimated time of 2100 seconds was used for S in the above formula, based on observed shutdown and reboot times from previous operational history. The elapsed wall clock time (not including shutdown and reboot) when the system utilization first dropped below 100% due to exhaustion of the input queue was 18546 seconds. The elapsed time when the last job finished was 20739 seconds. This gives T = 19643. The numerator of the formula for E is simply the sum of the rightmost column of Table 1, which is 7,437,476 CPU-seconds. In this test, 512 processors of the T3E were used, so that P = 512. The above formula then gives E = 0.74. We hope to have some test figures for some other vendor systems soon.

A Simulator for the ESP Test

To help evaluate the expected performance and sensitivity of the ESP test, a simulator has been developed to predict the runtime of a parallel system running this workload under various scheduling schemes. As part of this simulator, several simple scheduling algorithms, such as first-come-first-serve (FCFS) and best-fit-first (BFF), were implemented together with the option of using backfill schemes and checkpoint-restart facilities for running the full configuration jobs. The simulation results were used in the definition of the ESP test, in order to make sure that the resulting test did not exhibit pathological behavior. We have also used the simulator to estimate results of the ESP test on various system configurations, until more results can be obtained with the ESP on real computer systems.

So far, the most noticeable result obtained using the simulator has been its assessment of a checkpoint-restart facility. For example, the simulated runtime of the ESP test using a backfill BFF algorithm decreased from 7.3 hours to 4.8 hours when checkpoint-restart was employed. These times will be lower than actual test times, since they do not account for system costs such as I/O contention, swapping overheads, and processor fragmentation. Nonetheless, these results indicate that checkpoint-restart is likely to have a very significant impact on system effectiveness in a real operational setting. Indeed, as demonstrated in Figure 1, the NERSC T3E system achieved significant improvements in operational efficiency when checkpoint-restart was instituted.

We plan to do studies on real systems with the ESP system in order to confirm these simulator results.

Conclusion

System utilization and efficiency have been largely ignored to date in the high performance computing community, yet it has been shown that these factors can make a significant difference in the total throughput performance of the system. We have proposed a new benchmark test, the "ESP" test, that measures effective system performance" in a real-world operational environment. We hope that this test will be of use to system managers and will help to spur the community (researchers and vendors) to improve system efficiency.

Acknowledgement

We wish to thank the NERSC Division and its Department of Energy sponsors for their support of this activity, including research time, computer resources and management support. We also wish to thank several persons who assisted with this effort in the NERSC organization, including (in alphabetical order) Tina Butler, Michael Declerck, Brent Draney, Esmond Ng, Michael Welcome and Tammy Welcome.

References

1. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan and S. K. Weeratunga, "The NAS Parallel Benchmarks", International Journal of Supercomputer Applications, vol. 5, no. 3, Fall 1991, pg. 63-73.
2. Jack Dongarra, "Performance of Various Computers Using Standard Linear Algebra Software in a Fortran Environment", also known as the Linpack Benchmark Report. Available from http://www.netlib.org/benchmark/performance.ps.
3. Jay Blakeborough and Michael Welcome, "Scheduling Update on the T3E", Cray Users Group Conference CD-ROM Proceedings, May 1999.
4. Horst D. Simon, William T. C. Kramer and Robert Lucas, "Building the Teraflops/Petabytes Production Supercomputing Center", Fifth International Euro-Par Conference Proceedings, August 1999, pg. 61-77.
5. B. Ujfalussy, Xindong Wang, Xioguang Zhang, D. M. C. Nicholson, W. A. Shelton, G. M. Stocks, A. Canning, Yang Wang, and B. L. Gyorffy, "High Performance First Principles Method for Comlex Magnetic Properties", SC'98 CD-ROM Proceedings, Nov. 1998.