Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters

Rong Ge, Xizhou Feng, Kirk W. Cameron
Scalable Performance Laboratory
Department of Computer Science and Engineering
University of South Carolina, Columbia, SC 29208, USA
{ge, fengx, kcameron}@cse.sc.edu

ABSTRACT

Left unchecked, the fundamental drive to increase peak performance using tens of thousands of power hungry components will lead to intolerable operating costs and failure rates. High-performance, power-aware distributed computing reduces power and energy consumption of distributed applications and systems without sacrificing performance. Generally, we use DVS (Dynamic Voltage Scaling) technology now available in high-performance microprocessors to reduce power consumption during parallel application runs when peak CPU performance is not necessary due to load imbalance, communication delays, etc. We propose distributed performance-directed DVS scheduling strategies for use in scalable power-aware HPC clusters. By varying scheduling granularity we can obtain significant energy savings without increasing execution time (36% for FT from NAS PB). We created a software framework to implement and evaluate our various techniques and show performance-directed scheduling consistently saves more energy (nearly 25% for several codes) than comparable approaches with less impact on execution time (<5%). Additionally, we illustrate the use of energy-delay products to automatically select distributed DVS schedules that meet users' needs.

1. INTRODUCTION

Many high-end distributed systems use tens of thousands of power-hungry commercial components in clusters to achieve high performance. Power reduction and energy conservation are important in these systems for two major reasons: operating cost, and reliability.

Operating cost: Earth Simulator requires about 12 megawatts of peak power. Petaflop systems may require 100 megawatts of power [2], nearly the output of a small power plant (300 megawatts). At $100 per megawatt ($.10 per kilowatt), peak operation of this petaflop machine is $10,000 per hour. More conservative estimates of 20% peak operational power ($2,000 per hour) are still beyond the budget constraints of many high-end system centers. These estimates ignore the additional cost of dedicated cooling.

System reliability: Commodity components fail at an annual rate of 2-3% [21]. A petaflop system of about 12,000 nodes (CPU, DRAM, NIC, and disk) will sustain hardware failures once every twenty-four hours. According to formula based on the Arrhenius Law, component life expectancy decreases 50% for every 10° C (18° F) temperature increase. Reducing a component's operating temperature the same amount (consuming less energy) doubles the life expectancy.

To address operating cost and reliability concerns, large-scale systems are being developed with low power components. This strategy, used in construction of Green Destiny [22] and IBM BlueGene/L [4], requires changes in architectural design to improve performance. For example, Green Destiny relies on driving the Transmeta Crusoe processor [19] development while BlueGene/L uses a version of the embedded PowerPC chip modified with additional floating point support. In essence, the resulting high-end machines are no longer strictly composed of commodity parts – making this approach very expensive to sustain.

In contrast to Green Destiny and BlueGene/L, we seek to create distributed system middleware that exploits the inefficiencies common to many parallel applications and leverages commodity high-performance technologies. Our general power-aware approach is to use off-the-shelf Dynamic Voltage Scaling (DVS) technologies of emergent high-performance processors[1] in server-class systems and schedule cluster-wide power consumption to exploit scientific workload characteristics.

Power-aware clusters are distributed clusters containing components that have multiple power/performance modes (e.g. CPU's with DVS). The time spent within and transitioning to/from these power/performance modes determines the delay (cost in performance) and energy (cost in power, heat, etc.). We study distributed DVS scheduling techniques in power-aware clusters. External distributed DVS scheduling techniques are autonomous and control DVS power/performance modes in a cluster as processes separate from the application under study.

SC|05 November 12-18, 2005, Seattle, Washington, USA
©2005 ACM 1-59593-061-2/05/0011…$5.00

[1] DVS capabilities are now available in server-class architectures including Intel Xeon (SE7520 chipset) and AMD Opteron (Tyan s2882 board) dual processor nodes.

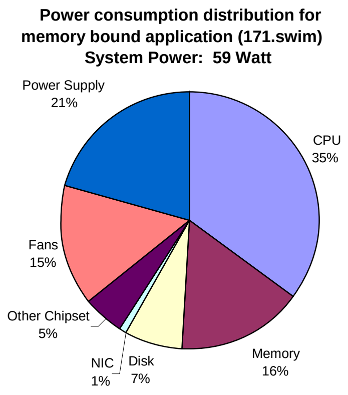


Figure 1: Under various workloads the CPU typically dominates server power consumption. For this memory bound code (swim from SPEC 2000) the CPU consumes 35% of total system power on a Pentium III-based node.

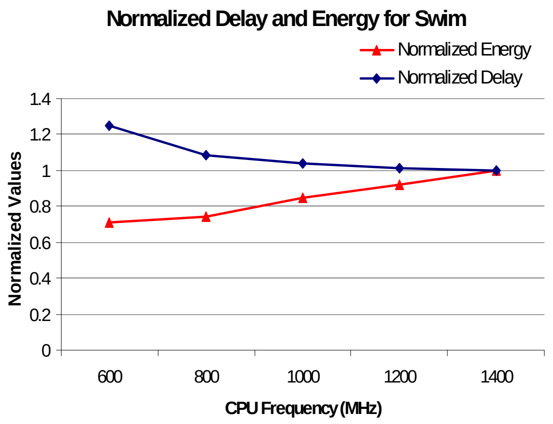


Figure 2: The energy-delay crescendo for swim shows the effect of application-dependent CPU slackness on (node) energy and performance measured at a single NEMO node. For swim, energy conservation can be achieved with (at times) reasonable performance loss.

External schedulers may be system-driven (e.g. the cpuspeed daemon) or user-driven (e.g. setting DVS from the command line). Internal distributed DVS scheduling techniques use source-level performance profiling to direct DVS power/performance modes with source-code instrumentation.

This paper is organized as follows: Section 2 provides an example to motivate our use of DVS to conserve cluster-wide energy while minimizing the impact on execution time. In Section 3, we provide a detailed description of the internal and external distributed DVS scheduling techniques used in our study. In Section 4, we describe our software framework to measure, profile, and control power consumption and performance in a distributed system. Section 5 gives the results of our detailed analysis of these scheduling techniques applied to the NAS Parallel Benchmarks. In Section 6 we describe how our approach differs from the work of others in this area. We close with some observations of trends in our data and future work.

2. MOTIVATING THE USE OF DVS IN HPC CLUSTERS

Dynamic Voltage Scaling (DVS) is a technology now present in high-performance microprocessors [1, 17]. DVS works on a very simple principal: decreasing the supply voltage to the CPU consumes less power. The dynamic power consumption (P) of a CMOS-based microprocessor is proportional to the product of total capacitance load (C), frequency (f), the square of the supply voltage (V^2), and percentage of active gates (A):

$$P \approx ACV^2 f \qquad (1)$$

Energy consumption (measured in joules) is reduced by lowering the average power (P_{avg}) consumed for some duration or delay ($D = t_2 - t_1$).

$$E = P_{avg} \times (t_2 - t_1) = P_{avg} \times D \qquad (2)$$

There are two compelling reasons for using DVS to conserve energy in HPC server clusters. The first reason is to exploit the dominance of CPU power consumption on system node (and thus cluster) power consumption. Figure 1 shows the breakdown of system node power obtained using direct measurement described by Feng et al [12]. This figure shows percentage of total system power for a Pentium III CPU (35%) under load. This percentage is lower (15%) but still significant when the CPU is idle. While the Pentium III can consume nearly 45 watts recent processors such as Itanium 2 consume over 100 watts and a growing percentage of total system power. Reducing the average power consumed by the CPU can result in significant server energy savings magnified in cluster systems.

The second reason for using DVS to conserve energy in HPC server clusters is to save energy without increasing execution time. DVS provides the ability to dynamically reduce power consumption. By reducing CPU power when peak speed is not needed (e.g. idle or slack periods) a DVS scheduling algorithm can reduce energy consumption. To minimize the impact on execution time we must ensure 1) supply voltage is reduced during CPU times when peak speed is not necessary, and 2) period duration outweighs voltage state transition costs[2].

Figure 2 shows the use of DVS on a single node to exploit CPU slack due to memory stalls. In this example we run swim from the SPEC 2000 benchmark suite on a DVS-enabled node at various fixed voltages shown as the resulting frequency[3] on the x-axis in increasing order. Lower frequency (i.e. lower voltage) means lower CPU performance. The values plotted on the y-axis are normalized to the highest (i.e. fastest) frequency respectively for energy and delay (execution time). This energy-delay "crescendo"

[2] In our AMD Opteron-based systems transition costs vary from 20-30 microseconds. Manufacturers set lower bounds (~10 microseconds) to achieve system stability following mode transitions.

[3] To be precise, DVS affects both voltage and frequency. Voltage and frequency are not independent as shown in Table 1. However for ease of discussion, we describe measurements in terms of the resulting frequency.

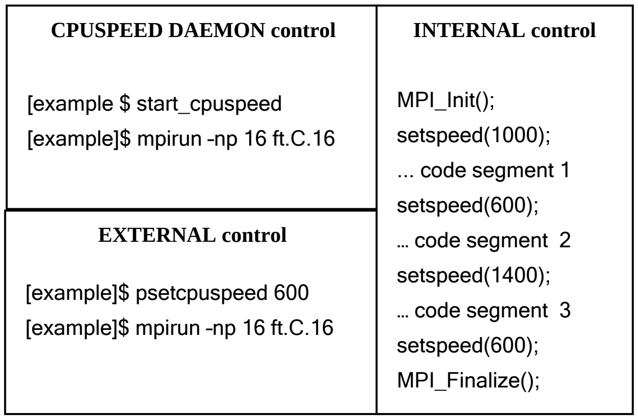
for swim shows when reducing CPU frequency (from right to left) the delay (execution time) increase varies from almost no increase at 1200 MHz to about 25% increase at 600 MHz. The corresponding total system energy decreases steadily with lower frequencies. Simply put, the memory stalls in swim produce enough slack periods for DVS to save energy (e.g. 8% at 1200 MHz) with almost no impact on execution time (<1%).

In this work, we study the tradeoffs of various DVS scheduling techniques designed to exploit CPU slack time in distributed systems. For parallel codes, idle CPU periods will be workload dependent and result from both memory and remote communication stalls. The rest of this paper describes our distributed DVS scheduling techniques, the software framework we designed and implemented to measure and control distributed DVS scheduling, and the results of our study.

3. DISTRIBUTED DVS SCHEDULING STRATEGIES

Now that we have established DVS as a viable approach to conserving energy while maintaining performance for HPC applications, we turn our attention to describing several approaches to schedule DVS transitions over the duration of a parallel code. Our goal in this section is not to explore every possible alternative in distributed DVS scheduling, but to provide detail on three techniques that differ in complexity and efficiency.

The scheduling techniques studied can be characterized as follows: 1) user- or system-driven, 2) internal or external to the application. The techniques can be evaluated by directly measuring the amount of total system energy consumed and the amount of execution time required to solution. In this section we discuss the techniques leaving discussion of our infrastructure for their evaluation to Section 4 and the results of our evaluation in Section 5. Figure 3 provides an overview of the three scheduling methods studied.


CPUSPEED DAEMON control

```
[example $ start_cpuspeed
[example]$ mpirun -np 16 ft.C.16
```

EXTERNAL control

```
[example]$ psetcpuspeed 600
[example]$ mpirun -np 16 ft.C.16
```

INTERNAL control

```
MPI_Init();
setspeed(1000);
... code segment 1
setspeed(600);
... code segment 2
setspeed(1400);
... code segment 3
setspeed(600);
MPI_Finalize();
```

Figure 3: Illustrations of the usage of three distributed DVS control strategies

3.1 CPUSPEED DAEMON

Strategy #1: Using the CPUSPEED Daemon

System-driven, external control of distributed DVS scheduling can be implemented as a system process or Daemon. The daemon we study is the CPUSPEED program included in the latest Fedora Core releases[4]. CPUSPEED schedules the DVS modes of a single node according to the CPU utilization information recorded by the system in the /proc directory of Linux. Other operating systems (e.g. Windows running on a laptop) provide comparable daemons for scheduling CPU, disk, and monitor power modes when the system is underutilized. These processes run autonomously and typically use saturation-based counters (or thresholds) and simple history-based information (e.g. CPU utilization) to migrate components between power modes.

Assuming a power aware node supports **m** operating points (voltage steps or frequencies) and the current operating point is **S**, the basic algorithm for CPUSPEED is as follows:

```
while( true )
{
   poll %CPU-usage from "/proc/stat"
   if %CPU-usage < minimum-threshold
      S = 0
   else if %CPU-usage > maximum-threshold
      S = m
   else if %CPU-usage < CPU-usage-threshold
      S = max( S-1, 0)
   else
      S = min( S+1, m)
   set-cpu-speed ( speed[S] )
   sleep (interval)
}
```

3.2 EXTERNAL

Strategy #2: Scheduling from the command-line

User-driven, external control can be implemented as a program invocation from the command line or as a system-call from a process external to the application. This is the approach used to save energy on a single node for the swim code shown in Figure 2. In the distributed version of this approach, the user synchronizes and sets the frequency for each node statically[5] prior to executing the application. For distributed applications that are memory/communication bound or imbalanced applications with large amounts of CPU slack or idle time, this approach is simple and effective. Performance profiling can be used to determine the amount of time the application spends stalled on the CPU for a given node configuration. Individual nodes can then be set to different DVS speeds appropriate to their share of the workload.

The process of DVS scheduling using external control is as follows: First we run a series of microbenchmarks to determine the effect of DVS on common types of code blocks including computationally intensive, memory intensive and communication intensive sequences. Next, we profile the performance of the application under study as a black box. We then determine which power mode settings are appropriate for the entire application running on a single node. Prior to execution, we set the individual nodes accordingly.

[4] See http://carlthompson.net/Software/CPUSpeed

[5] Dynamic settings are more appropriate for internal control from within the application (discussed next).

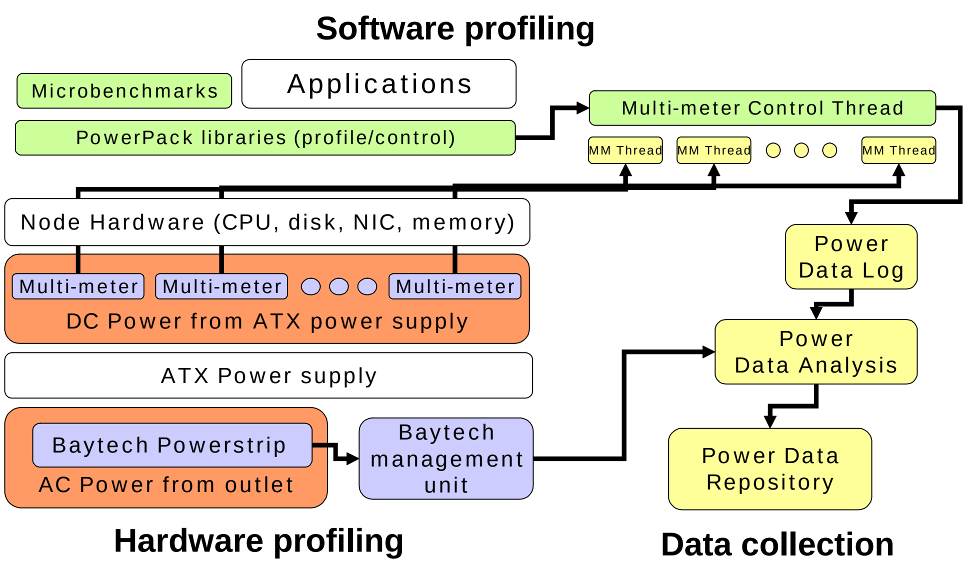


Figure 4: The PowerPack Framework. This software framework is used to measure, profile, and control several power-aware clusters including the NEMO cluster under study. Measurements are primarily obtained from the ACPI interface to the batteries of each node in the cluster and the Baytech Powerstrips for redundancy. The PowerPack libraries provide an API to control the power modes of each CPU from within the applications. Data is collected and analyzed from the Baytech equipment and the ACPI interface.

3.3 INTERNAL

Strategy #3: Scheduling within application

User-driven, internal control can be implemented using an API designed to interface with the power-aware component in the node. By controlling DVS from within an application, we can control the granularity of DVS mode transitions. The appropriate level of granularity depends on the amount of slack time and the overhead for mode transitions. For some codes with intensive loop-based computation, transitions between power modes around basic blocks may be appropriate. For other codes, function-level granularity may be more useful. At the extreme end, we can resort to external scheduling at node granularity (see Section 3.2).

Application-level control requires an API. We created such an API as part of our PowerPack framework discussed in Section 4. The insertion of API DVS control commands can be implemented by a compiler, middleware, or manually.

The process of DVS scheduling using internal API control is as follows: First we run a series of microbenchmarks to determine the effect of DVS on common types of code blocks including computationally intensive, memory intensive and communication intensive sequences. Next, we profile the performance of the application under study at a fine granularity identifying code sequences that share characteristics with our microbenchmarks. We then determine which power mode settings are appropriate for a given code sequence and insert the appropriate API calls around the code blocks. For now we do this manually. As part of future work we plan to integrate this into a compiler or run-time tool.

Figure 3 provides an example using each of the three strategies described. In the rest of this document, we use CPUSPEED DAEMON to refer to strategy #1, EXTERNAL to refer to strategy #2, and INTERNAL to refer to strategy #3. Using CPUSPEED DAEMON, users execute their application after the daemon is running. Using EXTERNAL, users determine a suitable operating frequency and set all the nodes to this operating point[6] (such as 600MHz in the example in Figure 3) before executing the application. Using INTERNAL, users insert DVS function calls into the source code, and execute the re-compiled application. When either external or internal scheduling is used, CPUSPEED must be turned off.

4. EXPERIMENTAL FRAMEWORK

Our experimental framework is composed of five components: experimental platform, performance and energy profiling tools, data collection and analysis software, microbenchmarks, and metrics for analyzing system power-performance.

4.1 NEMO: Power-aware Cluster

To better understand the impact of DVS technologies on future high performance computing platforms with DVS, we built a scalable cluster of high-performance, general purpose CPU's with DVS capabilities. Our experimental framework is shown in Figure 4. It interacts with NEMO, a 16-node DVS-enabled distributed system[7], Baytech power management modules and a data workstation.

---

[6] For now we focus on a homogeneous implementation of the EXTERNAL scheduling algorithm. Heterogeneous (different nodes at different speeds) is straightforward but requires further profiling which is actually accomplished by the INTERNAL approach.

[7] We use this system prototype to compare and contrast the scheduling policies discussed. Our techniques are general and equally applicable to emergent server-based clusters with DVS enabled dual AMD Opterons and Intel Xeon processors. This cluster was constructed prior to the availability of such nodes to the general public. We are currently building a cluster of dual AMD Opteron's with DVS capabilities which became available in fourth quarter 2004.

NEMO is constructed with 16 Dell Inspiron 8600 laptops connected by 100M Cisco System Catalyst 2950 switch. Each node is equipped with a 1.4 GHz Intel Pentium M processor using Centrino mobile technology to provide high-performance with reduced power consumption. The processor includes on-die 32KB L1 data cache, on-die 1 MB L2 cache, and each node has 1 GB DDR SDRAM. Enhanced Intel SpeedStep technology allows the system to dynamically adjust the processor among five supply voltage and clock frequency settings given by Table 1. The lower bound on SpeedStep transition latency is approximately 10 microseconds according to the manufacturer [18].

Table 1: Operating points for the Pentium M 1.4GHz processor

Frequency	Supply voltage
1.4GHz	1.484V
1.2GHz	1.436V
1.0GHz	1.308V
800MHz	1.180V
600MHz	0.956V

We installed open-source Linux Fedora Core 2 (version 2.6) and MPICH (version 1.2.5) on each node and choose MPI (message passing interface) as the model of parallel computation. We use CPUFreq as the interface for application-level control of the operating frequency and supply voltage of each node.

4.2 Power, energy and performance profiling

For redundancy and to ensure correctness, we use two independent techniques to directly measure energy consumption.

The first direct power measurement technique is to poll the battery attached to the laptop for power consumption information using ACPI. An ACPI smart battery records battery states to report remaining capacity in mWh (1mWh=3.6Joules). This technique provides polling data updated every 15-20 seconds. The energy consumed by an application is the difference of remaining capacity between execution beginning and finishing when system is running on DC battery power. To ensure reproducibility in our experiments, we do the following operations prior to all power measurements:

1) Fully charge all batteries in the cluster;
2) Disconnect (automatically) all laptops from wall outlet power remotely;
3) Discharge batteries for approximately 5 minutes to ensure accurate measurements;
4) Run parallel applications and record polling data.

The second direct power measurement technique uses specialized remote management hardware available from Bay Technical (Baytech) Associates in Bay St. Louis, MS. With Baytech proprietary hardware and software (GPML50), power related polling data is updated each minute for all outlets. Data is reported to a management unit using the SNMP protocol. We additionally use this equipment to connect and disconnect building power from the machines as described in the first technique.

To correlate the power profile and performance profile, we also generate profiles of tested applications automatically by using an instrumented version of MPICH. We do application performance and energy profiling separately because the overhead incurred by event tracing and recording.

4.3 PowerPack Software

While direct measurement techniques are collectively quite useful, it was necessary to overcome two inherent problems to use them effectively. First, these tools may produce large amounts of data for typical scientific application runs. Second, we must coordinate power profiling across nodes and hardware polling rates within a single application.

To overcome these difficulties, we created a software tool suite called PowerPack. PowerPack performs automating power measurement data collection and analysis in distributed systems. PowerPack also includes several portable libraries for (low-overhead) timestamp-driven coordination of power measurement data and DVS control at the application-level using system calls. ACPI is obtained and coordinated using our libraries libbattery.a. Lastly, we created software to filter and align data sets from individual nodes for use in power and performance analysis and optimization. The data in this paper is primarily obtained using our ACPI-related libraries; however data is verified using the Baytech hardware.

4.4 Power-performance microbenchmarks

We measure and analyze results for a series of microbenchmark codes (part of our PowerPack tool suite) to profile the memory, CPU, and network interface energy behavior at various static DVS operating points. These microbenchmarks are grouped into three categories:

1) Memory-bound microbenchmark
2) CPU-bound microbenchmark
3) Communication-bound microbenchmark

We leave disk-bounded microbenchmark for future study, though disk-bound applications will provide more opportunities to DVS for energy saving. We provide detailed discussions of our microbenchmark techniques in a related paper [15].

4.5 Energy-performance efficiency metrics

When different operating points (i.e. frequency) are used, both energy and delay vary even for the same benchmark. A fused metric is required to quantify the energy-performance efficiency. Brooks et al suggest using EDP (Energy-Delay Product) for high-end workstation and ED2P (Energy-Delay-Squared Product) for high performance server [6]. Cameron et al propose weighted ED2P metrics for DVS-enabled power aware cluster [7]. To ensure greater emphasis on performance, more weight on the delay factor is required.

In this work, we use ED2P ($E \times D^2$) and ED3P ($E \times D^3$) to choose “optimal” operating point (i.e., the CPU frequency that has the minimum ED2P or ED3P value for given benchmarks) in DVS scheduling for power-aware clusters. ED2P is proportional to Joule/MIPS^2, and ED3P is proportional to Joule/MIPS^3. Since ED3P metric puts more performance constraint than ED2P metric, smaller performance loss is expected for scheduling with ED3P in contrasting to scheduling with ED2P. In this work, both energy and delay are normalized with the values at the highest frequencies.

5. EXPERIMENTAL RESULTS AND DISCUSSION

This section presents our experimental results on three DVS scheduling strategies for the NAS parallel benchmarks (NPB) [3]. The benchmarks, which are derived from computational fluid applications, consist of five parallel kernels (EP, MG, CG, FT and IS) and three pseudo-applications (LU, SP and BT). These eight benchmarks feature different communication patterns and communication to computation ratios. We note experiments as XX.S.# where XX refers to the code name, S refers to the problem size, and # refers to the number of nodes. For example, LU.C.8 is the LU code run using the C sized workload on 8 nodes. In all our figures, energy and delay values are normalized to the highest CPU speed (i.e. 1400 MHz). This corresponds to energy and delay values without any DVS activity.

To ensure accuracy in our energy measurements using ACPI, we collected data for program durations measured in minutes. In some cases we used large problem sizes to ensure application run length was long enough to ensure accurate measurements. In other cases we iterate application execution. This ensures the relatively slow ACPI refresh rates (e.g. 15-20 seconds) accurately record the energy consumption of the battery for each node. Additionally, we repeated each experiment at least 3 times or more to identify outliers.

5.1 CPUSPEED DAEMON Scheduling

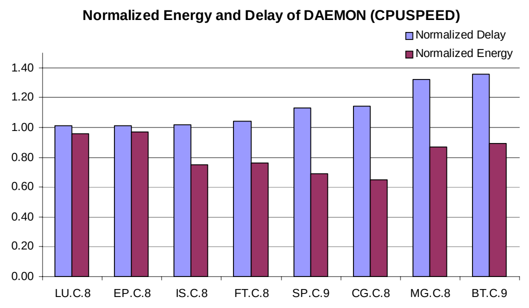


Figure 5: Energy-performance efficiency of NPB codes using CPUSPEED version 1.2.1. The results are sorted by normalized delay. Normalized delay is total application execution time with DVS divided by total application execution time without DVS. Values < 1 indicate performance loss. Normalized energy is total system energy with DVS divided by total system energy without DVS. Values < 1 indicate energy savings.

Figure 5 shows NAS PB results using CPUSPEED daemon controlled scheduling on our distributed power-aware cluster. We evaluate the effect of two versions of CPUSPEED: one is version 1.1 included in Fedora 2 and the other is version 1.2.1 included in Fedora 3. In version 1.1, the default minimum CPU speed transition interval value is 0.1 second; in version 1.2.1, the default interval value has been changed to 2 seconds. Since we have observed that CPUSPEED version 1.1 always chooses the highest CPU speed for most NPB codes without significant energy savings [15], only results of the improved CPUSPEED 1.2.1 are shown in Figure 5.

The effects of CPUSPEED vary with different codes. For LU and EP, it saves 3~4% energy with 1~2% delay increase in execution time. For IS and FT, it reduces 25% energy with 1~4% delay. For SP and CG, it reduces 31~35% energy with 13~14% delay increase. However, for MG and BT, it reduces 21% and 23% energy at the cost of 32% and 36% delay increase.

The original version of CPUSPEED 1.1 was equivalent to no DVS (our 1400 MHz data point) since threshold values were never achieved. CPUSPEED version 1.2.1 improves energy-performance efficiency for scientific codes significantly by adjusting the thresholds. We intend to study the affects of varying thresholds for applications that perform poorly even under the improved version in future work. Overall, CPUSPEED 1.2.1 does a reasonable job of conserving energy. However, for energy conservation of significance (>25%) 10% or larger increases in execution time are necessary, which is not acceptable to the HPC community. The history-based prediction of CPUSPEED is the main weakness of the CPUSPEED DAEMON scheduling approach. This motivates a study of scheduling techniques that incorporate application performance profiling in the prediction of slack states.

5.2 EXTERNAL Scheduling

In this section, we examine coarse-grain, user-driven external control which assumes users know the overall energy-performance behavior of an application but treat the internals of the application as a black box.

Section 4 describes the steps necessary to create a database of microbenchmark information for use in identifying DVS settings appropriate to an application. Applications with communication/-computation or memory/computation ratios that match micro-benchmark characteristics allow a priori selection of DVS settings. Here, our goal is to analyze this DVS scheduling approach for the power mode settings in our system.

Table 2 gives raw figures for energy and delay for all the frequency operating points available on our system over all the codes in the NAS PB suite. As is evident, such numbers are a bit overwhelming to the reader. Furthermore, selecting a “good” frequency operating point is a subjective endeavor. For instance, BT at 1200 MHz has 2% additional execution time (delay) with 7% energy savings. Is this “better” or “worse” than BT at 1000 MHz with 4% additional execution time and 20% energy savings? Such comparisons require a metric to evaluate. For this type of comparison, we use the ED3P ($\boldsymbol{ED^3}$) metric to weight performance as significant compared to energy savings.

Figure 6 shows the energy-performance efficiency of NPB benchmarks using external control with ED3P ($\boldsymbol{ED^3}$) metric. This figure is obtained as follows: For each benchmark, compute the $\boldsymbol{ED^3}$ value at each operating point using corresponding normalized delay and normalized energy, and use the operating point which has the smallest $\boldsymbol{ED^3}$ value as the scheduling point thereafter. If two points have the same $\boldsymbol{ED^3}$ value, choose the point with best performance. The effect of external DVS scheduling shown in Figure 6 reduces energy with minimum execution time increase and selects an operating frequency that is application dependent – thus overcoming the weakness of CPUSPEED.

Table 2: Energy-performance profiles of NPB benchmarks

Only partial results are shown here. In each cell, the number on the top is the normalized delay and the number at the bottom is the normalized energy. The column “auto” means scheduling using CPUSPEED. The columns “XXX MHz” refer to the static external setting of processor frequency.

Code	CPU Speed					
	auto	600 MHz	800 MHz	1000 MHz	1200 MHz	1400 MHz
BT.C.9	1.36	1.52	1.27	1.14	1.05	1.00
	0.89	0.79	0.82	0.87	0.96	1.00
CG.C.8	1.14	1.14	1.08	1.04	1.02	1.00
	0.65	0.65	0.72	0.80	0.93	1.00
EP.C.8	1.01	2.35	1.75	1.40	1.17	1.00
	0.97	1.15	1.03	1.02	1.03	1.00
FT.C.8	1.04	1.13	1.07	1.04	1.02	1.00
	0.76	0.62	0.70	0.80	0.93	1.00
IS.C.8	1.02	1.04	1.01	0.91	1.03	1.00
	0.75	0.68	0.73	0.75	0.94	1.00
LU.C.8	1.01	1.58	1.32	1.18	1.07	1.00
	0.96	0.79	0.82	0.88	0.95	1.00
MG.C.8	1.32	1.39	1.21	1.10	1.04	1.00
	0.87	0.76	0.79	0.85	0.97	1.00
SP.C.9	1.13	1.18	1.08	1.03	0.99	1.00

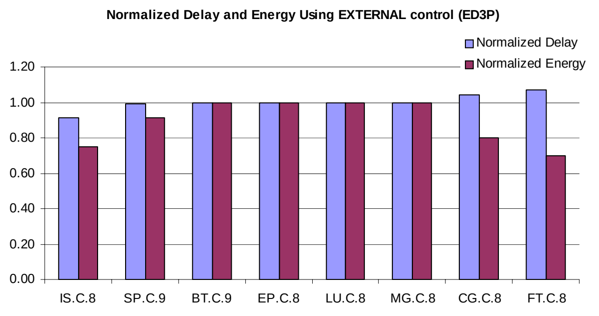


Figure 6: Energy-performance efficiency of NPB codes using EXTERNAL DVS control. ED3P is chosen as the energy-performance metric in this figure. The results are sorted by normalized delay.

The effects of external DVS scheduling can be classified in three categories:

1) Energy reduction with minimal performance impact. For FT, it saves 30% energy with 7% delay increase in execution time. For CG, it save 20% energy with 4% delay increase in execution time.

2) Energy reduction and performance improvement at the same time[8]. For SP, it saves 9% energy and also improves execution time by 1%. For IS, it saves 25% energy with 9% performance improvement.

3) No energy savings and no performance loss. BT, EP, LU, MG fall into this category.

Figure 6 uses ED3 as an energy-performance metric that favors performance to energy saving. We listed the energy and delay values for the frequency operating points that ED3P would select automatically for a user given a data set as shown in Table 2. If users allow slightly larger performance impact for more energy saving, ED2P (***ED***^{2}) or EDP (***ED***) could be selected as the energy-performance metric. Figure 7 shows the effects of ED2P metrics used with external DVS scheduling. The trend is the same as Figure 6, but the metric may select frequency operating points where energy savings have slightly more weight than execution time delays. For example, ED2P would select different operating points for FT corresponding to energy savings of 38% with 13% delay increase; for CG, it selects 28% energy with 8% delay increase. For SP, it selects 19% energy with 3% delay increase.

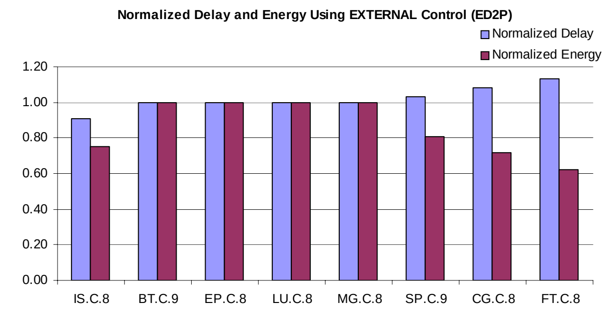


Figure 7: Energy-performance efficiency of NPB codes using EXTERNAL control. ED2P is chosen as the energy-performance metric in this figure. The results are sorted by normalized delay.

The benefits of external control are limited by three factors: 1) the application’s energy-performance crescendos using DVS; 2) the granularity of DVS control; and 3) reduced benefits for workload imbalance and heterogeneous distribution across the computing nodes.

[8] The results that IS and SP don’t achieve better performance at highest frequency are repeatable. Similar phenomena have also been observed for other benchmarks and by other researchers. The initial explanation is that message communication is not sensitive to frequency above certain threshold. Within a busy network, higher frequency may increase the probability of traffic collision and result longer waiting time for packet retransmission. However, this explanation requires to be verified through further analysis and experiments.

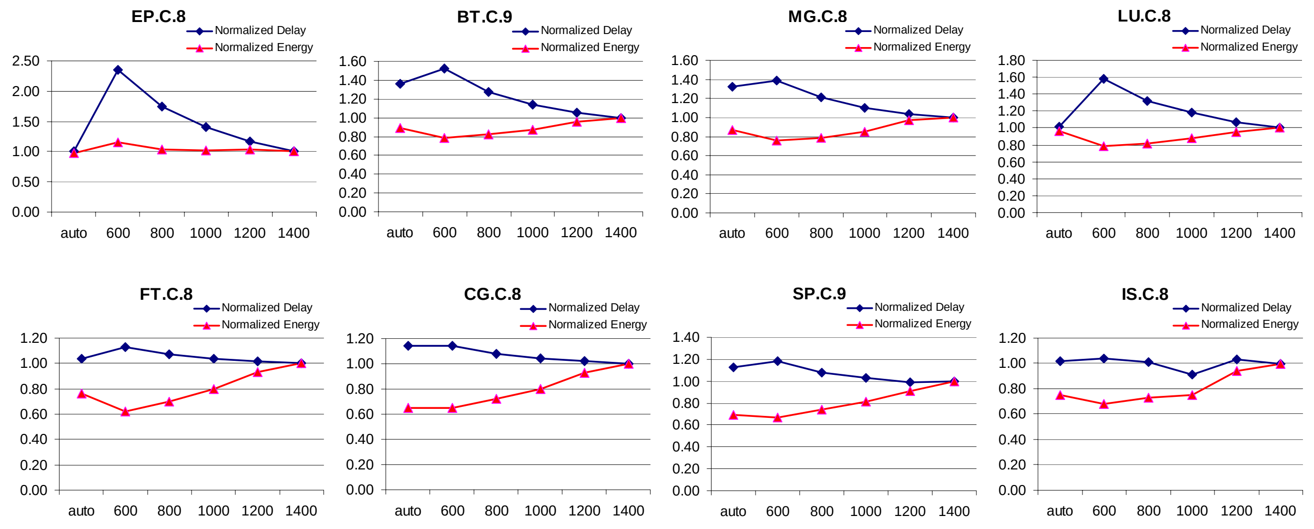


Figure 8: Energy-delay crescendos of NPB benchmarks. X-axis is CPU speed. Y-axis is the normalized value (delay and energy). The eight figures are grouped into four categories.

The first limitation determines whether DVS is useful for an application or not. Figure 8 shows graphically the energy-performance profiles under external DVS scheduling using energy-delay crescendos (see Section 2). These figures indicate that we can group the eight benchmarks into four categories:

Type I (EP): near zero energy benefit, linear performance decrease when scaling down CPU speed.

Type II (BT, MG and LU): near linear energy reduction and near linear delay increase, the rate of delay increase and energy reduction is about same.

Type III (FT, CG and SP): near linear energy reduction and linear delay increase, but the rate of delay increase is smaller than the rate of energy reduction.

Type IV (IS): near zero performance decrease, linear energy saving when scaling down CPU speed.

This classification matches the effects of external control shown in figure 6 and figure 7. In other words, the observed trends indicate basically that Type III and Type IV save energy while Type I and Type II do not save energy.

The second and third limitations are not shown in the figure but can be understood analytically: a real parallel application will consist of combinations of dependent computation modules which belong to two or more of the categories mentioned above. If we only schedule a single static CPU speed for all nodes during the whole execution, benefits obtained from Type III and Type IV will be compromised by the impact of Type I and Type II. Therefore, we need to use internal control to overcome this limitation. Internal control (discussed next) must consider application execution phases.

5.3 INTERNAL Scheduling

We use FT.C.8 and CG.C.8 as examples to illustrate how to implement internal scheduling for different workload and its effects and limitation. Each example starts with performance profiling and then the DVS scheduling strategy is derived by analyzing the profiles. The effects of the scheduler are verified with experimental results.

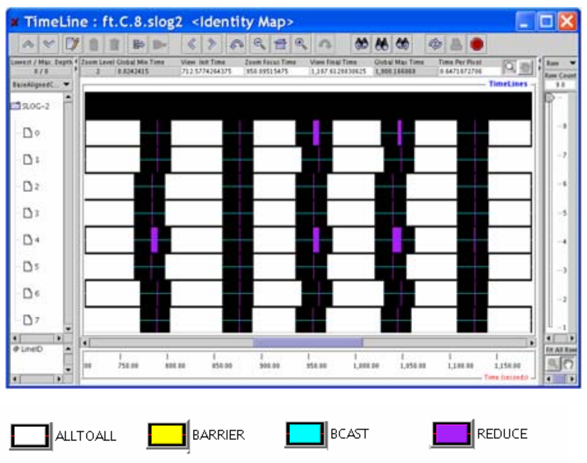


Figure 9: Performance trace of FT.C.8 using MPE tool provided with MPICH. The traces are visualized with Jumpshot.

5.3.1 *Internal scheduling for FT benchmark*

Performance Profiling: Figure 9 shows the performance profile of FT generated with MPICH trace utility by compiling the code with "–mpilog" option. The following observations are drawn from the profile.

1) FT is communication-bound and its communication to computation ratio is about 2:1.

2) Most execution time is consumed by all-to-all communication.

3) The execution time per iteration is sufficiently big such that the CPU speed transition overhead can be ignored.

4) The workload is almost balanced across all nodes.

Scheduling Design: based on the above observation we divide each iteration into all-to-all communication phases and other phases. Therefore the CPU is set to low speed for all-to-all communication phase and restored to high CPU speed thereafter. Figure 10 shows how DVS control is inserted into the source code.

```
…
call set_cpuspeed( low_speed)
call mpi_alltoall( … )
call set_cpuspeed( high_speed)
…
```

Figure 10: INTERNAL control for FT benchmark

Experiment Results: Figure 11 shows the energy savings and increase in execution time delays using internal scheduling. By choosing high_speed as 1400 MHz and low_speed as 600 MHz, internal scheduling can save 36% energy without noticeable delay increase. This is significant improvement over both external control and CPUSPEED. External control at 600MHz saves 38% energy but the cost is 13% delay increase. CPUSPEED saves 24% energy with 4% delay increase. This shows internal scheduling is preferred when the application contains obvious CPU-bound phases and non-CPU bounded phases and each phase lasts long enough to compensate for the CPU speed transition overhead.

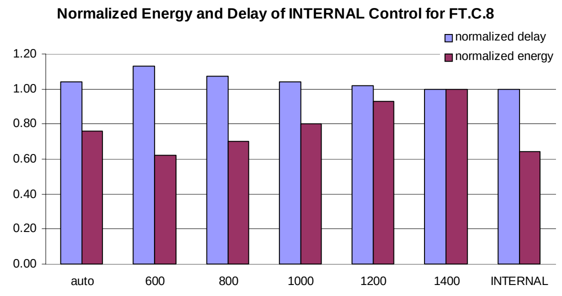


Figure 11: Normalized energy and delay of INTERNAL control, EXTERNAL scheduling and CPUSPEED. In INTERNAL control, high speed and low speed are set as 1400 and 600 MHz respectively. All EXTERNAL control's decisions (600MHZ-1400MHz) are given on the x-axis. CPUSPEED is shown as auto in this figure. Normalized delay is total application execution time with DVS divided by total application execution time without DVS. Values < 1 indicate performance loss. Normalized energy is total system energy with DVS divided by total system energy without DVS. Values < 1 indicate energy savings.

5.3.2 *Internal Control for CG benchmark*

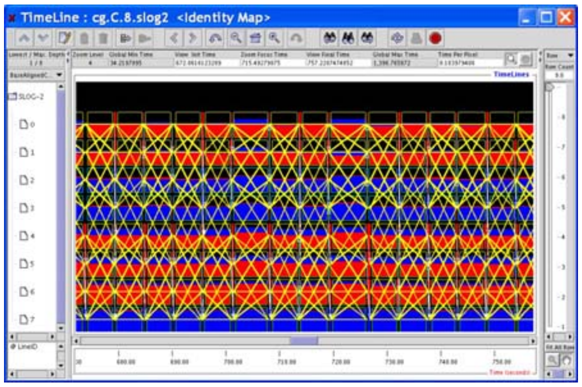
Performance Profiling: Figure 12 shows the performance profile of CG with the following observations:

1) CG is communication intensive and synchronizes all nodes within each cycle.

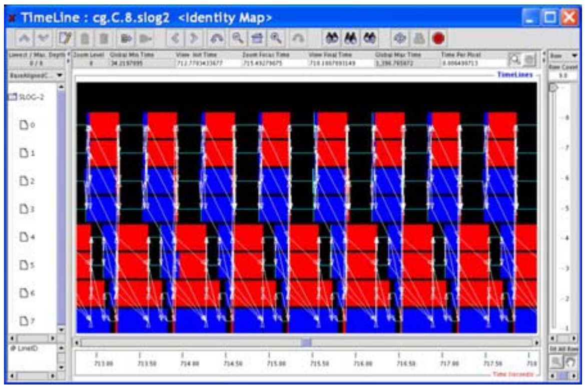
2) Wait and Send are major communication events.

3) The execution time of each cycle is relatively small, the message communications are frequent and CPU speed transition overhead can not be ignored.

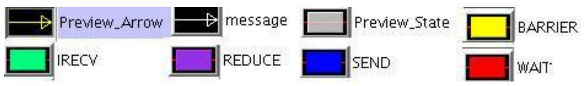
4) Different nodes exhibit different communication and computation behavior. Nodes 4-7 have larger communication-to-computation ratio compared to nodes 0-3.



(a) Profile visualized at iteration granularity



(b) Profile visualized at message granularity.



(c) The meaning of the logo in (a) and (b)

Figure 12: Performance trace of CG.C.8 using MPE tool provided with MPICH. The traces are visualized with Jumpshot.

Scheduling Decision: Based on the above observations, we found it challenging to improve power-performance efficiency with phased-based DVS scheduling. We have implemented two phase-based dynamic scheduling policies for CG: the first is to scale down CPU speed during communication; the second is to scale down CPU speed during MPI_Wait. Both phase-based DVS scheduling approaches increase both energy and delay (1~3%). According to the asymmetric performance behavior on each node, we can set different speeds for each execution node. The DVS controls are inserted as shown in Figure 13.

```
…
if ( myrank .ge. 0 .and. myrank .le. 3)
   call set_cpuspeed( high_speed)
else
   call set_cpuspeed( low_speed)
endif
…
```

Figure 13: INTERNAL control for CG benchmark

Experimental Results: The results from the experiments are shown in Figure 14. We provide results for two configurations: internal I which uses 1200 MHz as high speed and 800 MHz as low speed and internal II which uses 1000 MHz as high speed and 800 MHz as low speed. Experiments show that internal I saves 23% energy with 8% delay increase and internal II saves 16% energy with 8% delay increase. Both internal I and II scheduling for CG do not provide significant advantages over external scheduling at 800MHZ. This is reasonable since CG requires frequent synchronization and the external control aggregates gains and losses across all nodes.

Overall: Internal scheduling provides DVS control with finer granularity than external scheduling. Internal scheduling achieves better (or at least as good) energy-performance efficiency. FT shows the benefit of phased-based internal scheduling; CG shows the benefit of heterogeneous internal scheduling.

6. RELATED WORK

Dynamic Voltage Scaling has been studied for decades, mostly in the area of energy-constrained, low power, real time system and mobile system. Researchers have developed various DVS scheduling algorithms to save energy under timing deadlines [20, 23].

As low power technologies have migrated to most core components of high-performance systems including processor, disk, memory, network card [8, 9, 11], researchers have studied the effects of power-aware technologies on general purpose processors to conserve energy while maintaining performance.

Some work has also been accomplished in distributed systems[5, 8]. These studies focus on conserving energy in clusters of web servers. Energy is conserved by exploiting the characteristics of interactive workloads [16]. Tasks are scheduled and migrated to optimally conserve energy in data centers.

However, the differences between interactive workloads and scientific workloads require different power management strategies. Hsu and Kremer use compiler-directed dynamic voltage and frequency scheduling to exploit slackness for energy savings using non-interactive scientific workloads [15]. This is an important step towards power-aware high performance computing though it targets sequential applications on a single processor. Recently, several research groups have studied using DVS scheduling to improve the energy-performance efficiency in high performance computing. Cameron et al and Freeh et al have demonstrated significant energy savings can be achieved with minimum performance impact by exploiting the computing efficiencies in parallel scientific computing [7, 13, 14]. Chen et al suggested scaling down the CPU speed on nodes that are not in the critical path so that energy can be saved without performance penalty [10].

Our work is orthogonal to these studies. In this paper, though we provide further evidence that high-performance power-aware distributed computing is viable, we focus on implementing and analyzing various distributed DVS scheduling policies. Such policies are critical to automating middleware that alleviates users from thinking about power and energy consumption. Our results indicate given user-defined energy-performance efficiency metrics, our schedulers can reduce energy and guarantee performance. Our experiments all indicate that no single scheduling strategy fits all scientific codes.

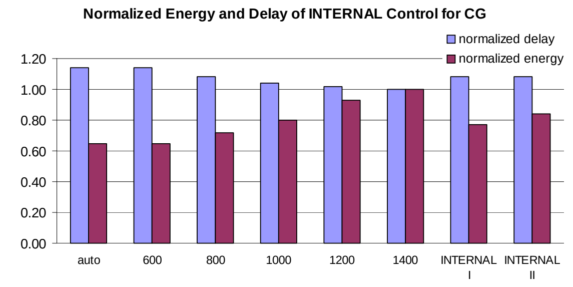


Figure 14: Normalized energy and delay of INTERNAL scheduling, EXTERNAL control and CPUSPEED scheduling for CG. For INTERAL I, high speed is 1200, and low speed is 800; for INTERNAL II, high speed is 1000 and low speed is 800.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have designed and implemented distributed DVS scheduling for power-aware clusters, and created a framework for application-level power measurement and optimization. We directly measure, analyze, and compare several DVS strategies on a power-aware cluster to conserve power while maintaining performance in scientific parallel applications.

Our results reinforce the recent discovery that significant amounts of energy can be saved in parallel scientific applications while maintaining performance. We achieved total energy savings as large as 36% with no negative impact on performance. However, we also showed that energy savings vary greatly with application, workload, system, and DVS strategy.

We identified the primary weaknesses in history-based scheduling techniques applied to scientific applications. We also showed that the most flexible scheduling technique (INTERNAL) does not always provide significant gains over simpler implementations such as EXTERNAL command line settings.

While this work brings us closer to middleware that can provide energy savings automatically with little user intervention, our techniques are largely manual and more work is needed to fully automate the process. Our future work includes attempting to improve automation and address the limitations of the various scheduling techniques. Our continuing goal is to improve energy savings while maintaining performance through better prediction methods more suitable to high-performance computing applications.

ACKNOWLEDGEMENTS
The authors would like to thank the National Science Foundation and the Department of Energy for sponsoring this work under grants NSF CCF-#0347683 and DOE DE-FG02-04ER25608 respectively.

REFERENCES

[1] AMD, "Mobile AMD Duron Processor Model 7 Data Sheet," AMD, 2001.

[2] D.H. Bailey, "21st Century High-End Computing," In invited Talk Application, Algorithms and Architectures workshop for BlueGene/L, 2002.

[3] 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 and High Performance Computing, vol. 5, pp. 63-73, 1991.

[4] BlueGene/LTeam, "An overview of the BlueGene/L supercomputer," Supercomputing 2002 Technical Papers, 2002.

[5] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kister, C. Lefurgy, C. Mcdowell, and R. Rajamony, "The Case For Power Management in Web Servers," in Power Aware Computing, R. Graybill and R. Melhem, Eds. IBM Research, Austin TX 78758, USA.: Klewer Academic, 2002.

[6] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P.W. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," IEEE Micro, vol. 20, pp. 26-44, 2000.

[7] K.W. Cameron, R. Ge, X. Feng, D. Varner, and C. Jones, "POSTER: High-performance, Power-aware Distributed Computing Framework," presented at Proceedings of 16th International Conference on High Performance Computing and Communications (SC 2004), 2004.

[8] E.V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," presented at Proceedings of the 17th International Conference on Supercomputing, 2003.

[9] S. Chandra, Wireless network interface energy consumption implications of popular streaming formats. Multimedia Computing and Networking (MMCN'02), vol. 4673. San Jose, CA: The International Society of Optical Engineering, 2002.

[10] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing Power with Performance Contraints for Parallel Sparse Applications," presented at The First Workshop on High-Performance, Power-Aware Computing, Denver, Colorado, 2005.

[11] X. Fan, C.S. Ellis, and A.R. Lebeck, "The synergy between power-aware memory systems and processor voltage scaling," Department of Computer Science Duke University, Durham TR CS-2002-12, 2002.

[12] X. Feng, Rong Ge, Kirk Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," presented at 19th International Parallel and Distributed Processing Symposium (IPDPS 05), Denver, CO, 2005.

[13] V.W. Freeh, D.K. Lowenthal, F. Pan, and N. Kappiah, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," presented at 10th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP), 2005.

[14] V.W. Freeh, D.K. Lowenthal, R. Springer, F. Pan, and N. Kappiah, "Exploring the Energy-Time Tradeoff in MPI Programs," presented at 19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS), Denver, Colorado, 2005.

[15] R. Ge, X. Feng, and K.W. Cameron, "Improvement of Power-Performance Efficiency for High-End Computing," presented at 1st Workshop on High-Performance, Power-Aware Computing (HPPAC 2005), in conjunction with IPDPS'2005, Denver, Colorado, 2005.

[16] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," presented at ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI'03), San Diego, CA, 2003.

[17] Intel, "Developer's manual: Intel 80200 Processor Based on Intel XScale Microarchitecture.," 1989.

[18] Intel, "Intel Pentium M Processor datasheet," 2004.

[19] D. Laird, "Crusoe Processor Products and Technology," Transmeta, 2000.

[20] J.R. Lorch and A.J. Smith, "PACE: A new approach to dynamic voltage scaling," Ieee Transactions on Computers, vol. 53, pp. 856-869, 2004.

[21] D.A. Patterson and J.L. Hennessy, Computer Architecture: A quantitative approach, 3rd ed. San Fancisco, CA: Morgan Kaufmann Publishers, 2003.

[22] M.S. Warren, E.H. Weigle, and W.-C. Feng, "High-Density Computing: A 240-Processor Beowulf in One Cubic Meter," presented at IEEE/ACM SC2002 Conference, Baltimore, Maryland, 2002.

[23] A. Weissel and F. Bellosa, "Process Cruise Control-Event-Driven Clock Scaling for Dynamic Power Management," presented at Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002), Grenoble, France, 2002.