Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications

Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah,
Rob Springer, Barry L. Rountree, and Mark E. Femal, *Member*, *IEEE*

Abstract—Although users of high-performance computing are most interested in raw performance, both energy and power consumption have become critical concerns. One approach to lowering energy and power is to use high-performance cluster nodes that have several power-performance states so that the energy-time trade-off can be dynamically adjusted. This paper analyzes the energy-time trade-off of a wide range of applications—serial and parallel—on a power-scalable cluster. We use a cluster of frequency and voltage-scalable AMD-64 nodes, each equipped with a power meter. We study the effects of memory and communication bottlenecks via direct measurement of time and energy. We also investigate metrics that can, at runtime, predict when each type of bottleneck occurs. Our results show that, for programs that have a memory or communication bottleneck, a power-scalable cluster can save significant energy with only a small time penalty. Furthermore, we find that, for some programs, it is possible to *both* consume less energy *and* execute in less time by increasing the number of nodes while reducing the frequency-voltage setting of each node.

Index Terms—High-performance computing, power-aware computing.

1 Introduction

High-performance computing (HPC) tends to push performance at all costs. Unfortunately, the "last drop" of performance tends to be the most expensive; that is, the last 10 percent increase in performance requires a disproportionally large amount of resources. The Earth Simulator, one of the world's fastest supercomputers, consumes 7 MW of power [19]. It is unlikely that supercomputing centers can continue increasing consumption of resources. In particular, energy consumption—and the resultant heat dissipation—is becoming an important limiting factor; reducing energy saves money and increases reliability, among other things.

As a result, low-power high-performance clusters have been developed to stem the ever-increasing demand for energy. Such systems improve the energy efficiency of nodes. In particular, Green Destiny [72] consumes about three times less energy per unit performance than the Accelerated Strategic Computing Initiative (ASCI) Q machine. However, because Green Destiny uses a slower (and cooler) microprocessor, ASCI Q is about 15 times faster per node (200 times overall) [72]. A reduction in performance by such a factor is likely unreasonable from the point of view of many HPC programmers.

We believe one should choose a path between these two extremes using a high-performance commodity microprocessor that has both frequency and voltage scaling. Each frequency-voltage pair provides a power-performance point that we call a *gear*. It is possible to reduce power consumption without a significant increase in execution time because an increase in CPU frequency generally results in a smaller increase in application performance. The reason for this is that the CPU is not always the bottleneck resource. Therefore, increasing frequency also increases CPU stalls—which are usually due to the CPU blocking while waiting for data from either the memory subsystem or another node.

This paper investigates the trade-off between energy and performance (execution time) for HPC applications on a real small-scale power-scalable cluster. Our cluster has as its nodes AMD-64 processors that support frequency and voltage scaling. This paper investigates the trade-off between energy and performance (execution time) for both serial and parallel HPC programs. Our results illustrate an *energy-time trade-off*: By this, we mean that decreasing energy is possible at the cost of increased time. Not every energy-time trade-off is desirable, as some offer little energy savings and large time penalties. However, more than half of our tests show a savings equal to or better than the penalty (for example, 10 percent less energy and 10 percent more time), and some are much better. For example, we found that the single-node version of *cg* from the NASA Advanced Supercomputing (NAS) suite allowed 20 percent less energy to be used while only increasing execution time by 3 percent. With others, such as *ep*, there was essentially no energy savings and a large time penalty. Additionally, we found that, in some cases, one can save energy *and* time by executing a program on more nodes at a slower gear rather than on fewer nodes at the fastest gear.

We also present and analyze two simple metrics, *misses per operation* (MPO) and *slack*, that can predict this energy-time trade-off. We argue that these metrics will allow system software designers to determine when to use frequency scaling and which gear to use.

- *V.W. Freeh, F. Pan, N. Kappiah, and M.E. Femal are with the Department of Computer Science, North Carolina State University, Raleigh, NC. E-mail: {vwfreeh, fpan2, nkappia, mefemal}@ncsu.edu.*
- *D.K. Lowenthal, R. Springer, and B.L. Rountree are with the Department of Computer Science, The University of Georgia, Athens, GA. E-mail: {dkl, springer, rountree}@cs.uga.edu.*

Manuscript received 8 June 2005; revised 15 Feb. 2006; accepted 13 June 2006; published online 9 Jan. 2007.
Recommended for acceptance by A. Sivasubramaniam.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0285-0605.
Digital Object Identifier no. 10.1109/TPDS.2007.1026.

The rest of this paper is organized as follows: Section 2 describes related work and Section 3 describes our methodology. We show results for single and multiple nodes, respectively, in Sections 4 and 5. Section 6 discusses categorizing applications for the purpose of choosing the proper energy gear using online metrics. Section 7 provides discussion of three aspects of this work. Finally, Section 8 summarizes the paper.

2 Related Work

There has been a voluminous amount of research performed in the general area of energy management. In this section, we describe some of the closely related research. We divide the related work into two categories: server/desktop systems and mobile systems.

2.1 Server/Desktop Systems

Several researchers have investigated saving energy in server-class systems. The basic idea is that, if there is a large enough cluster of such machines, such as in hosting centers, energy management can become an issue. In [7], Chase et al. illustrate a method to determine the aggregate system load and then determine the minimal set of servers that can handle that load. All other servers are transitioned to a low-energy state. A similar idea leverages work in cluster load balancing to determine when to turn machines on or off to handle a given load [63], [62]. The policy proposed in [60], as well as several new polices, was investigated in [15].

Such work shows that power and energy management is critical for commercial workloads, especially Web servers [5], [47]. Another approach [30] minimizes energy for a cooperative Web server running on heterogeneous cluster nodes by directing requests to be serviced by the appropriate node; different nodes can consume vastly different amounts of power. Additional approaches that have been taken include dynamic voltage scaling (DVS) [66] and request batching [14]. The work in [66] applies real-time techniques to Web servers in order to conserve energy while maintaining quality of service.

Our work differs from most prior research because it focuses on HPC applications and installations, rather than commercial ones. A commercial installation tries to reduce cost while servicing client requests. On the other hand, an HPC installation exists to speed up an application, which is often highly regular and predictable. One approach is to save energy in an application-specific way; Chen et al. [8] used this approach for a parallel sparse matrix application. Another HPC effort that addresses the memory bottleneck is given in [34]; however, this is a purely static approach. Also, Feng et al. [20] developed a measurement infrastructure for power-aware HPC programs on a cluster of laptops and performed experiments on NAS programs. In [37], a generic evaluation infrastructure that makes use of SimPoint [67] is described; this infrastructure combines simulation and direct measurement. Another approach that can be used for HPC and is independent of the application can be found in [43]; performance counters are used to estimate IPC dynamically and choose the correct gear. Finally, metrics for power-aware HPC are described in [36].

In server farms, disk energy consumption is also significant. One study of four energy conservation schemes concludes that reducing the spindle speed of disks is the only viable option for server farms [6]. DRPM is a scheme that dynamically modulates the speed of the disk to save energy [27], [29]. Several have investigated RAID systems specifically. In [28], it was shown that tuning the traditional RAID parameters such as stripe size was the key to optimizing both energy and time. On the other hand, Li and Wang [48] looked at new scheduling and caching schemes, and Pinheiro and Bianchini [61] migrated popular data to a subset of disks so the others can be transitioned to a low power state. The work in [78] generalizes [61] in the sense that the disks can run at different speeds, and the data are migrated to an appropriate disk. Another approach is to improve cache performance—if many consecutive disk accesses are cache hits, the disk can be profitably powered down until there is a miss; this is the approach taken by Zhu et al. [79]. An alternative is to infer the access pattern based on inspection of the program counter and shut down the disk accordingly [24]. A final approach is to try to aggregate disk accesses in time. A compiler/runtime approach using this was designed and implemented in [31], and a prefetching approach in [58].

There are also a few HPC clusters designed with energy in mind. One is BlueGene/L [1], which uses a "system on a chip" to reduce energy. Another is Green Destiny [72], which uses low-power Transmeta nodes. A related approach is the Orion Multisystem machines [54], though these are targeted at desktop users.

Additionally, an analysis of energy efficiency and operating points in [53] shows that the most energy-efficient gear does not always result in the lower performance point. Some approaches estimate power consumption using performance counters [4], [40], [24]. In a high-performance processor, a dominant part of power consumption is due to exploiting instruction level parallelism (ILP). As the ILP increases, the processor becomes busier, and there is a subsequent increase in power. Explorations of IPC and performance are done in [9], [70], and [55]. In addition, IPC is utilized in [39] for an architectural simulator for both dynamic energy efficiency and temperature management. Kotla et al. use IPC as a means of predicting future performance to schedule frequency changes in a server cluster [43]. Section 6 discusses an alternative metric (MPO) for characterizing the criticality of the CPU.

This paper is partly based on our previously published works. A study of single-node applications can be found in [57]; however, it explores energy-time trade-off in mobile systems. In [22], we study the energy-time trade-off in message passing interface (MPI) programs and develop a simulation infrastructure for larger clusters. We have also examined the energy saving potential by using multiple energy gears in MPI programs [23]. In a separate work, we develop techniques for increasing throughput by improving the power/energy efficiency of nodes in data centers [17], [18].

2.2 Mobile Systems

There is also a large body of work in saving energy in mobile systems; most of the early research in energy-aware

computing was on these systems. Here, we detail some of these projects.

ECOSystem [77] attempts to implement a power management system without the need to rewrite application software using the *currency* model. The case for a closer relationship between the operating system and power management is further explored by Vahdat et al. [69]. This includes a case for treating energy as a first-class resource in operating systems. Perhaps the best endorsement of OS-controlled power consumption comes from the Advanced Configuration and Power Interface (ACPI) standard [10]. It is an evolution of several existing methods including basic input/output system (BIOS) power management, the Advanced Power Management (APM) API, and a smart battery interface.

In general, the goal of the OS is to reduce energy, given a multiprogrammed workload; although performance is important, on a laptop, it is fairly likely that energy is the first concern. Our approach differs in that we are concerned with reducing system energy for a single HPC program while sacrificing little performance.

2.2.1 Individual Devices

Many have worked on saving energy in different devices, such as the CPU, disk, memory, and network. Many modern processor architectures allow different frequency-voltage settings. This work developed into DVS [21], [26], [59], [64], which has come to mean the simultaneous changing of clock speed and voltage to reduce power consumption. Typically, DVS optimizes the $\text{energy} \times \text{delay}$ or $\text{energy} \times \text{delay}^2$ [51], [25] product. This creates a system that more efficiently uses energy, but is still powerful and responsive. In order to do this, DVS must accurately *predict* upcoming load. A poor prediction either consumes too much energy or causes too much delay. In this work, we investigate the relationship between frequency-voltage and execution time.

2.2.2 Disk

Disks consume a large percentage of energy on some mobile architectures. Many have studied disk spindown to save energy (for example, [32], [13], [74], [3], and [49]). In general, the idea is to determine when there is a large time period in which there are no disk requests and transition to a lower energy level. Cooperative I/O allows applications to *defer* I/O operations in order to save energy [73]. In this paper, we do not consider disk energy.

2.2.3 Memory and Network

In some architectures, individual memory banks and network cards have multiple energy states. Works on transitioning these to lower energy states include [11] and [46]. The idea is to place data intelligently in banks so that some banks will not be accessed. In some devices, the network card has multiple energy states; methods to save network energy include [44], [76], and [45]. In this paper, we do not consider memory or network power.

The primary distinction between these projects and ours is that energy saving is typically the primary concern in mobile devices. In HPC applications, performance is still the primary concern. Hence, our work is orthogonal to these efforts.

3 Methodology

This section describes our experimental methodology. In our single-node tests, we studied programs from three different benchmark sets: NAS, System Performance Evaluation Cooperative (SPEC) integer, and SPEC floating point. The NAS suite is a popular HPC benchmark. It consists of scientific benchmarks, including application areas such as sorting, spectral transforms, and fluid dynamics. In contrast, the SPEC integer benchmarks are nonscientific applications that are CPU and/or memory intensive. The SPEC floating point benchmarks are a mixture of both scientific and nonscientific programs. For example, *mesa* and *facerec* are nonscientific graphics programs, whereas *swim* and *mgrid* are well-known scientific benchmarks. In our multinode tests, we studied the NAS MPI suite as well as some of the ASCI benchmarks [2]. We used all valid configurations up to nine nodes. Presumably, such mature benchmarks have been thoroughly analyzed and are well written (for example, see [75]). Thus, they are not unrealistically memory or communication bound (which would make frequency scaling appear better than it will typically be in practice).

As stated earlier, this study uses, as a reference example, a cluster of 10 nodes, each equipped with a frequency and voltage-scalable AMD Athlon-64. All single-node tests are run on one of the nodes. We run each program at each available energy gear: 800 to 2,000 MHz in steps of 200 MHz. The voltage, which ranges from 1.5 to 1.0 V, is reduced in each gear. Each node has a 1-Gbyte main memory, a 128-Kbyte L1 cache (split), and a 512-Kbyte L2 cache. The nodes are connected by a 100 megabit-per-second network. In this paper, we control the CPU power (by shifting gears) and measure overall system energy. This is effective in saving energy because the CPU—a major power consumer—uses less power.

For each program, we measure the execution time and energy consumed for an application at a range of energy gears. The execution time is the elapsed wall clock time. The energy consumed by the entire system is calculated by measuring the power consumption of each node using a WattsUp Pro meter [52] attached to each node at the wall outlet. This value is integrated over time to determine the energy used. Because we attach a meter to each node, for multinode tests, the total energy consumption by all systems can be easily and accurately obtained by calculating the sum of each individual node's energy consumption.

Table 1 shows idle power and active power ranges for all gears of the AMD-64. The active power ranges are determined using single-node data collected in Section 4. The idle power is determined with CPU in the C1 state using the *halt* command. In Linux, the CPU enters the C1 state when the operating system determines that the system is idle. We can see that the difference in active power consumption between highest and lowest frequencies is about 40 percent, whereas the difference in idle power is approximately 15 percent.

The time to shift between any two gears is at most 500 microseconds and generally less than half that [16]. Fig. 1 shows that the overhead of gear shifting is acceptably small for the range of shifting frequencies shown. The

TABLE 1
Idle and Active (Total System) Power for Each AMD-64 Node

Frequency (MHz)	*Voltage* (V)	*Power* (W)	
		idle	active
2000	1.5	59.6	98.8–108.6
1800	1.4	57.0	86.5–95.2
1600	1.35	55.9	80.1–88.1
1400	1.3	54.8	74.3–81.9
1200	1.2	53.4	67.1–75.4
1000	1.1	52.0	62.2–69.6
800	1.0	51.0	56.8–63.5

energy consumption and elapsed time of the *lu* benchmark from the NAS suite are shown for a single node in the cluster. The plots show these values for various shifting frequencies. For example, the leftmost points show the energy and time when shifting between the top two gears every 8 seconds (0.125 shifts/seconds). The extreme horizontal lines show the time and energy values when the sequential *lu* program is run exclusively in the top two gears, which does not depend on shifting rate. The middle horizontal line is the analytic average value of the top two gears. All shifting programs spend approximately half the time in each gear. Therefore, the theoretical expectation is that the shifting values are the average of the two fixed values. The figure shows that the shifting points are all within 1 percent of the average value (note that the extreme lines differ by 5 and 8 percent in time and energy, respectively).

Section 4 shows the single-node results, which illustrate the energy-time trade-off due to the memory bottleneck. Then, Section 5 describes the multiple node results, which illustrate the effect of the communication bottleneck as well as the energy-time trade-off when the number of nodes increases.

4 Single-Node Results

Because we studied a large number of programs from three benchmark suites, space constraints do not permit the presentation of all the individual results. Therefore, we divide single-node results into two parts. First, we discuss the overall results. Then, we look at a few representative applications in detail. Complete individual application results can be found in [56].

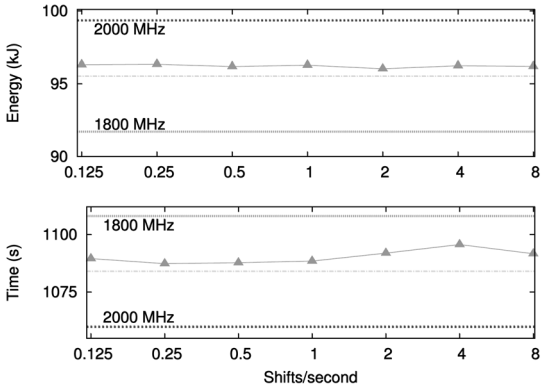

Fig. 1. Examining the effects of shifting gears on *lu*.

4.1 Overall Results

All of our tests show that, for a given program, using the fastest gear takes the least time. A slower gear takes more time and uses less power. However, this may or may not result in a decrease in overall system energy.

If the decrease in power exceeds the increase in time, a slower gear uses less energy. This is the case for all of the programs we tested, where one of the slower gears results in the least energy consumed. However, it is conceivable that the time increase could exceed the power decrease, so the fastest gear would also consume the least energy.

Fig. 2 plots the normalized aggregate results for each program set on one node (NAS, SPEC INT, and SPEC FP). The x-axis plots the gear in terms of frequency from highest to lowest. There are three lines showing time, power, and energy. All values are normalized to those of the fastest gear; thus, all lines begin at 1 on the left-hand side. The increasing line is the elapsed time, and the strictly decreasing line is the average power consumption. The energy consumption, which is the product of the power and time, initially decreases and then rises. In each benchmark suite, the power consumption decreases with frequency to approximately the same degree. There is little difference in power consumption because all programs in all sets are simple "batch" programs: no communication, little I/O delay, and so forth. While the power always decreases, the time delay at the slowest gears is so great that the energy savings, if any, is small. However, we notice that all of the programs have an energy-time trade-off: There is an energy

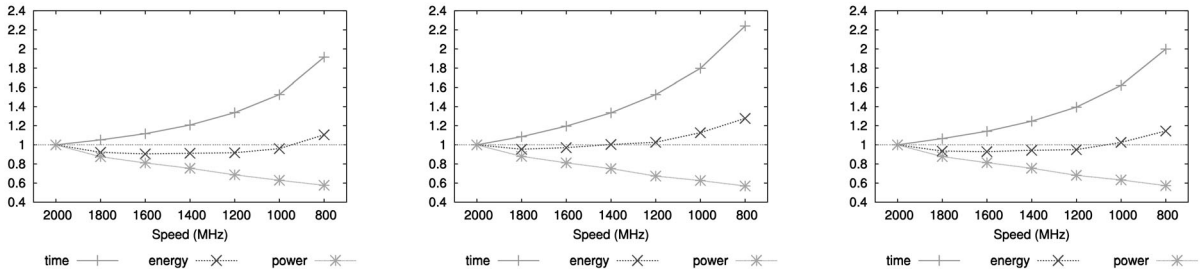

Fig. 2. Aggregate plots of all programs in each set. (a) NAS. (b) SPEC INT. (c) SPEC FP.

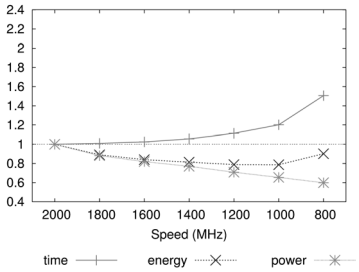

Fig. 3. Aggregate plot of the three NAS programs that have least CPU bound.

savings (and a time delay) at a reduced frequency. Complete results are shown in [56].

For the NAS programs, the time and energy diverge from 1 about equally for higher frequencies. This means the energy savings is approximately equal to the time delay. For example, at 1,600 MHz, the energy used and time taken are 91 and 112 percent of full, respectively. The SPEC sets also show an increase in time, but show little decrease in energy. The programs in the SPEC suites are much more CPU bound than NAS, as discussed in Section 6. Although the SPEC sets are common benchmark suites, many of the programs are not representative of HPC applications. On the other hand, all the NAS programs are simplified HPC programs with large data sets; therefore, a typical HPC application is more likely to resemble a NAS program than a SPEC program.

Some programs, especially those that are CPU bound, do not benefit from CPU scaling, Therefore, Fig. 3 shows the results of a subset of the NAS suite that is least CPU bound (*cg*, *lu*, and *sp*). In this subset, the energy used and time taken are 86 and 105 percent of full, respectively, at 1,600 MHz. This result indicates there is significant benefit to CPU scaling, but it must be applied judiciously.

4.2 Detailed Results

In this section, we analyze six programs in detail that represent the best and worst in terms of energy-time trade-off from each program set. The data for a program are presented using a scatter plot of energy versus time. The total system energy consumed at each gear is plotted on the y-axis, and the total execution time is plotted on the x-axis. The higher of the two points uses more energy, and the further right of two points takes more time. Therefore, a near-vertical slope indicates an energy savings with little time delay between adjacent gears, whereas a horizontal slope indicates a time penalty and no energy savings.

The programs shown in Fig. 4 have the best energy-time trade-off in each set: NAS (*cg*), SPEC INT (*mcf*), and SPEC FP (*art*). In these *vertical* applications, the execution time advantage of the fastest gear is small. However, the energy *penalty* for this best performance is large. Consider, for example, the *cg* benchmark in Fig. 4a. Setting the gear to 1,400 MHz yields a 3 percent increase in execution time compared to the fastest, whereas the corresponding decrease in energy consumption is 20 percent.

Next, we examine how a vertical energy-time shape occurs. Our experience shows that programs use essentially the same number (within 1 percent) of microoperations regardless of the gear. This is the case for all SPEC and NAS programs. However, the number of cycles that an execution takes can change, especially in the vertical applications. For example, consider the *mcf* application at the two fastest gears (2,000 and 1,800 MHz), in which the performance gain is less than 1 percent. Using the slower gear with a clock rate that is 90 percent of the highest, the execution has 90 percent as many cycles (approximately 5.0 to 4.5 trillion). Because the number of microoperations does not change, the microprocessor throughput, in microoperations per cycle (UPC), increases as the frequency decreases. The additional cycles in the faster gear do not perform useful work. This indicates that the CPU is not the performance bottleneck. Below, we examine this and, not surprisingly, determine that memory is the bottleneck.

On the other hand, Fig. 5 shows the programs that do not exhibit an energy penalty for the ultimate performance. Instead, in these programs, the fastest gear is nearly the lowest energy consumed. We call these *horizontal* programs. The time penalty (increase in execution time) is more than 10 percent for all three programs, which is nearly the same as the increase in the CPU cycle time (11 percent). Thus, these programs are highly CPU bound. The minimum energy is always at 1,800 MHz, and the savings is less than 4 percent. The reason the energy decreases is that the power consumption decreases by 13 to 14 percent, which is more than the time increases. Therefore, the last 10 percent of performance requires more than 10 percent more power.

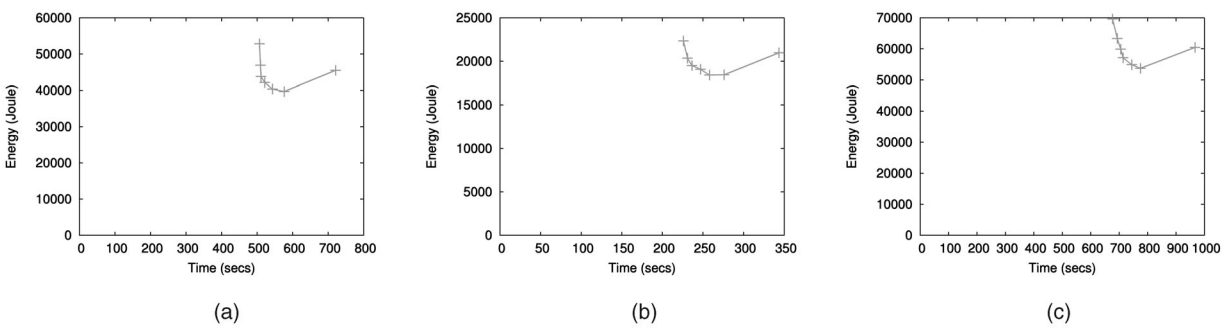

Fig. 4. Best energy-time trade-off in each set. (a) *cg*. (b) *mcf*. (c) *art*.

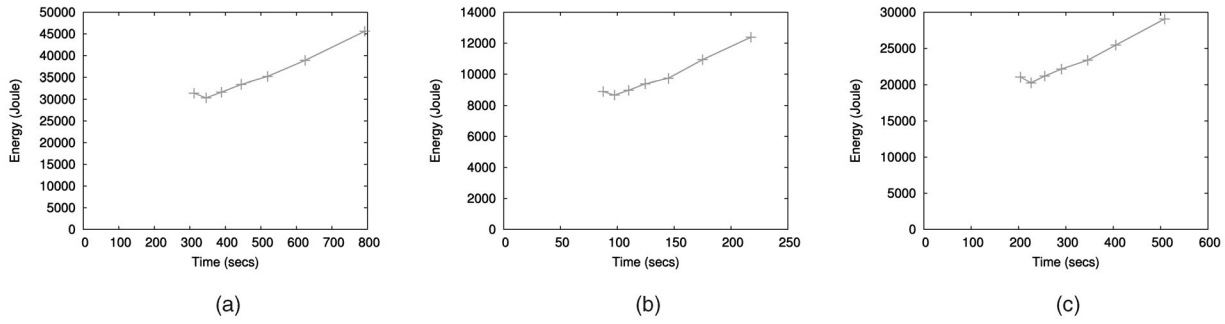

Fig. 5. Worst energy-time trade-off in each set. (a) *ep*. (b) *crafty*. (c) *sixtrack*.

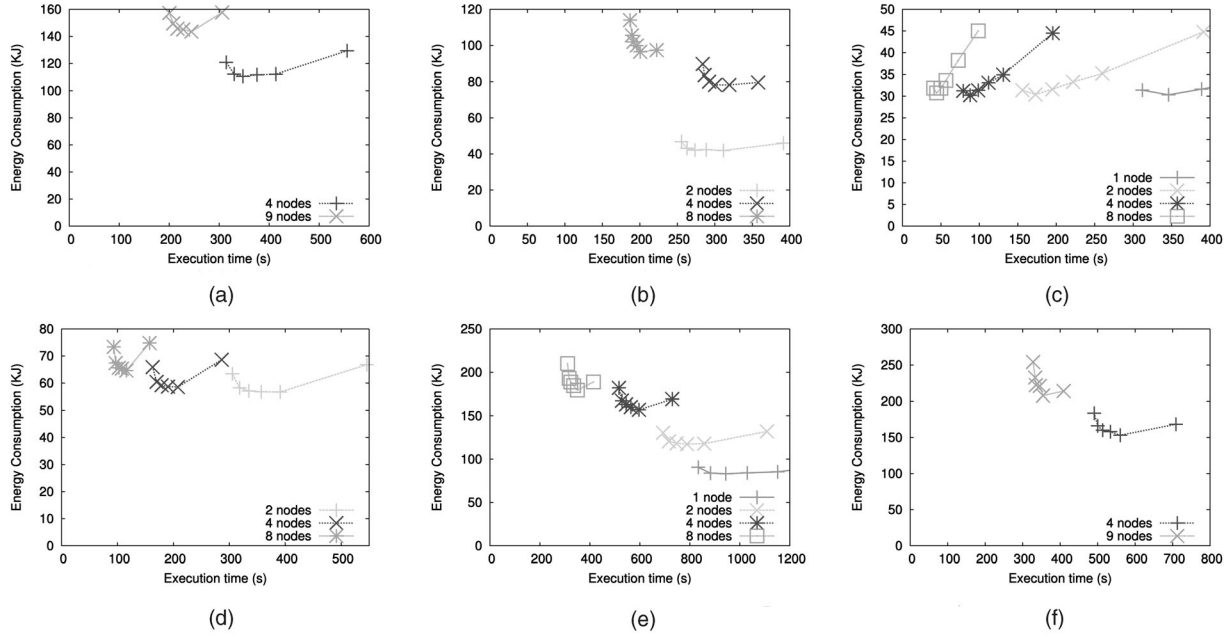

Fig. 6. Energy consumption versus execution time for NAS benchmarks for one to eight or nine nodes. (a) *bt* B—poor speedup. (b) *cg* B—poor setup. (c) *ep* B—perfect speedup. (d) *lu* B—good speedup. (e) *mg* B—poor/good speedup. (f) *sp* B—poor speedup.

5 Multiple-Node Results

Section 4 investigated the energy-time trade-off on a single node. This section studies the effect of distributed programs. Fig. 6 shows results from six NAS programs. (We do not show *is* or *ft* because they do not have any parallel speedup on our cluster.) Each graph has the same general layout as in Figs. 4 and 5, except that it shows the results from multiple experiments: two, four, and eight nodes (or four and nine nodes in the case of *bt* and *sp*). It also plots the one-node results from Section 4, but in most cases, the data are to the right of the window of time shown. The energy plotted is the cumulative energy of all nodes used.

5.1 Overall Results

Before discussing the results, we describe the possible layouts of these graphs. First, for a fixed number of nodes, the shape of the curve depends on the memory *and* communication bottlenecks. This is because, in a distributed program, not only might a processor wait for the memory subsystem, but, at times, it might also block while awaiting a message. In either scenario, the CPU is not on the critical path, and it is more efficient to execute in a lower energy gear.

Second, consider the possible effects when comparing an experiment with twice the number of nodes $(2P)$ versus one with P nodes. The following possibilities exist. Note that we do not consider the case where the time on $2P$ nodes is *larger* than on P nodes.

1. The curve for $2P$ nodes can lie completely above and to the left of the curve for P nodes (more energy, less time). Each point on the $2P$ node curve lies above all points on the P node curve. This case occurs when the program achieves **poor speedup** on $2P$ nodes compared to P nodes.
2. The point that represents the fastest energy gear for $2P$ nodes can be to the left of, and at or below, the corresponding point on the curve for P nodes. This case occurs when the program achieves **perfect or superlinear speedup** on $2P$ nodes compared to P nodes.
3. The curve for $2P$ nodes can lie to the left of the curve for P nodes, but not completely above or below the fastest gear point for P. This is the most interesting case. While the program executes faster and consumes more energy in the fastest gear on $2P$ nodes

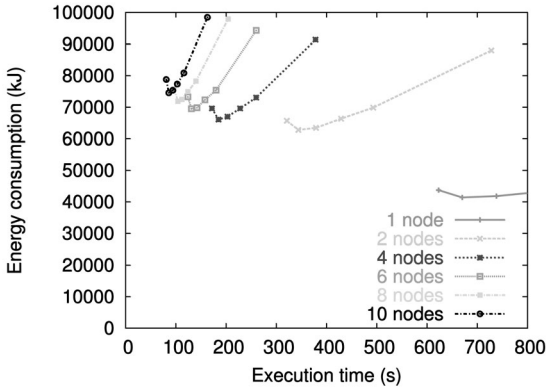

Fig. 7. Energy consumption versus execution time for Jacobi iteration on two, four, six, eight, and 10 nodes.

than on P nodes, there is a lower gear at $2P$ nodes that has less energy consumption than the fastest gear point at P nodes. Therefore, it is possible to achieve better execution time *and* lower energy consumption by running in a slower gear on $2P$ nodes than in a faster gear on P nodes. There is no energy-time trade-off between these points because one point dominates the other in both energy and time. This case occurs when **speedup is good** (that is, not superlinear and not poor) and there are a significant number of main memory accesses (so that scaling down the processor has only a slightly detrimental effect).

We describe each of the cases in turn below.

Case 1: Poor Speedup. Fig. 6 offers several examples of Case 1. In particular, this case is illustrated in *bt*, *sp*, and *mg* from two to four nodes, and *cg* both from two to four and four to eight nodes (the two-to-four case for *cg* actually slowed down). Suppose a given supercomputer cluster was restricted to a certain amount of power consumption or heat dissipation. This limit could be represented by a horizontal line that a program must remain below. The most desirable point would be the leftmost (fastest) one under this limit. For programs in this case, the horizontal line will intersect at most one of the curves.

Case 2: Perfect or Superlinear Speedup. Fig. 6 does not contain an example of superlinear speedup. However, *ep*, which gets almost perfect speedup, comes extremely close to illustrating this case. Power consumption doubles when the number of nodes doubles. Because the time is cut in half, the total energy consumed is the same. With superlinear speedup, the energy consumption decreases as nodes are added. When speedup is perfect or superlinear, there is no energy-time trade-off.

Case 3: Good Speedup. Fig. 6 shows several examples of this case. First, consider *lu* at four and eight nodes. Gear 4 on eight nodes uses approximately the same energy as the fastest gear on four nodes but executes 50 percent more quickly. The fastest gear executes 72 percent faster on eight nodes than on four nodes but uses 12 percent more energy. This case illustrates an additional choice not available in a conventional cluster, which only supports either of the fastest gear options (four or eight nodes). So, a user must trade off a performance increase against an energy increase. With a power-scalable cluster, the user can select a slower gear on eight nodes, which may offer better performance for the same energy consumption. Thus, a user of a power-scalable cluster has two dimensions to explore: 1) number of nodes and 2) processor performance gear. In Case 3, the user may be able to get better performance by using more nodes, with each node executing at a lower energy gear.

Next, Fig. 7 plots data for the Jacobi iteration application. This application is shown because it can run at any number of nodes, unlike the NAS benchmarks. The figure shows energy-time curves on five configurations: two, four, six, eight, and 10 nodes. Because this application gets good speedup (1.9, 3.6, 5.0, 6.4, and 7.7), each adjacent pair of curves falls under Case 3. For example, executing in the second or third fastest gear on six nodes results in the program finishing faster *and* using less energy than using the fastest gear on four nodes.

5.2 Detailed Results

So far, we have discussed the results in absolute terms. It is also useful to look at the relative shapes of the energy-time curves. Fig. 8 shows the curves for three of the NAS programs relative to the fastest gear. Therefore, the fastest gear is always plotted at (1, 1). These plots are presented because they illustrate the three cases described above. Fig. 8a shows that, for *ep*, increasing the number of nodes has little effect on the *relative* energy or time. This is because *ep* has an almost perfect speedup. Consequently, the program executes nearly identically on any number of nodes.

The next case is illustrated in Fig. 8b. This program is *bt*, which speeds up poorly because it has a large communication component. Although an increase in CPU cycle time increases computation time, it has little effect on the communication

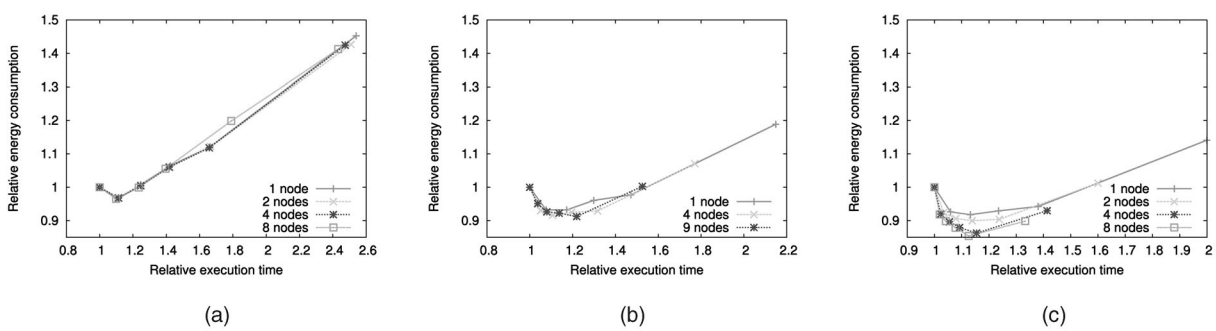

Fig. 8. Relative energy versus time. (a) *ep* B. (b) *bt* B. (c) *mg* B.

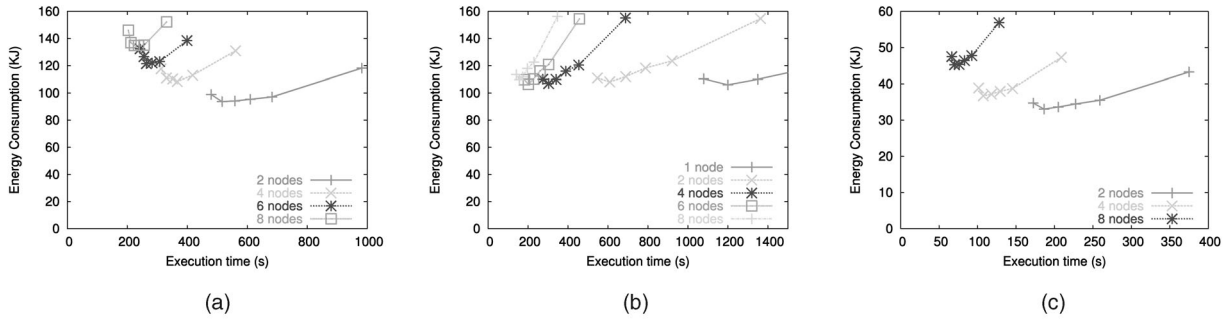

Fig. 9. Energy-time trade-off of some applications. (a) Aztec—poor/good speedup. (b) Sphot—perfect speedup. (c) Sweep3d—poor setup.

time. Therefore, the increase in execution time due to frequency reduction becomes less significant as the computation to communication ratio decreases. This effect is shown in Fig. 8b, as the curve compresses horizontally as the number of nodes increases. For example, the rightmost (slowest) point decreases from 2.15 to 1.76 to 1.52.

The last case is illustrated by *mg*, which gets good speedup (Fig. 8c). This example shows how the curve becomes more vertical as parallelism increases, indicating that the time penalty decreases. Both *mg* and *bt* have similar memory behavior, so the difference in these shapes must be due to communication. The important factor is the relative amounts of computation (executing) and communication (blocked) time. The gear setting primarily affects the active portion. Because *bt* has significant idle/blocked time (33 and 50 percent on four and nine nodes, respectively), the gear setting has a small effect on *bt*. However, *mg* has a much smaller amount of idle time (10, 13, and 24 percent on two, four, and eight nodes, respectively); therefore, the gear setting makes a more pronounced difference.

5.3 Additional Applications

Fig. 9 shows the energy-time trade-off of more modern benchmarks from the ASCI suite [2]. The ASCI programs typically execute for hours if not days on 96 processors or more, depending on inputs. They are designed to push the limits of current high-performance clusters and aid in the assessment of requirements for the next generation of clusters. We chose data set sizes appropriate for our cluster and truncated iterations to shorten the duration of the tests. There are a different number of tests in each figure because the programs do not support the same set of node configurations.

The figure shows that these applications have varying behavior and that each "maps to" one of the NAS programs. For example, *sphot* has near-perfect speedup, so there is almost no energy penalty for using more nodes; this is similar to *ep* from the NAS suite. On the other hand, the shape of the graph for *aztec* resembles that of *lu*. There is good but not perfect speedup, and it can be beneficial, in both time and energy, to use more nodes at a lower gear per node. Finally, *sweep3d* is similar to *mg* in that it attains reasonable speedup, but each individual curve is not quite as steep (though the steepness increases with the number of nodes due to communication overhead).

6 Categorizing Applications

The previous sections have shown opportunities for energy saving in HPC programs on both a single node and multiple nodes. As mentioned earlier, the challenge for system software designers is to exploit these opportunities when they are available—and only when they are available. For example, programs like *cg* should be run at a lower gear on a modest number of nodes, whereas programs like *ep* should be run at fastest gear on as many nodes as possible.

The question is stated as follows: Which metrics should be used to categorize programs into equivalence classes? As explained in Sections 4 and 5, the two key relevant characteristics are memory performance and speedup. We therefore investigated metrics to quantify these. Because we are primarily interested in inferring which gear to use dynamically, we prefer to determine effective *online* metrics —as many have done in the past (for example, [65]).

First, we investigated a metric termed β, introduced by Hsu and Kremer in [34] and applied to HPC by Hsu and Feng [35], which quantifies the extent to which the CPU is on the critical path. It compares the *application slowdown* to the *CPU slowdown*. Application slowdown is $\frac{T-T_{max}}{T_{max}}$, where T and T_{max} are the execution times at the corresponding frequencies. The CPU slowdown is $\frac{f_{max}-f}{f}$, where f is a given frequency and f_{max} is the highest frequency (2,000 MHz in this case). The criticality of the CPU, β, is defined as

$$\beta = \frac{\frac{T-T_{max}}{T_{max}}}{\frac{f_{max}-f}{f}} = \frac{\frac{T}{T_{max}}-1}{\frac{f_{max}}{f}-1}.$$

Theoretically, if a program is completely CPU bound, then T would increase by the amount of increase in cycle time, $T = \frac{f_{max}}{f} \cdot T_{max}$, in which case $\beta = 1$. On the other hand, if a program is completely independent of the CPU, then $T = T_{max}$ and $\beta = 0$. For these programs, the largest difference we observed between any two β values (same application but different frequencies) is 5 percent.

The NAS suite has an average β of 0.40, and half of those programs are less than 0.50. On the other hand, the SPEC suites have a β of 0.59 and 0.71 for the integer and floating point benchmarks, respectively. Furthermore, only one INT program and one-third of the FP programs have a β less than 0.50. This indicates that the NAS programs are less CPU bound than SPEC.

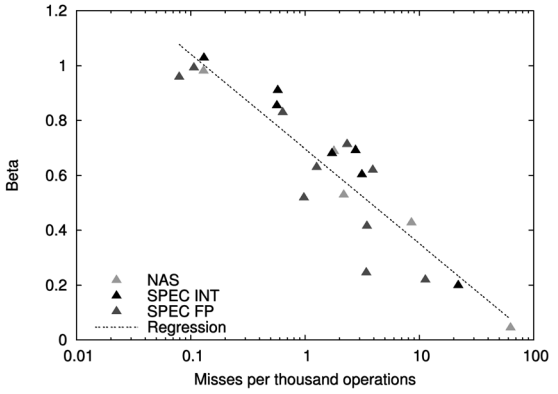

Fig. 10. MPO versus β for all single-node applications on AMD-64.

In general, β is an excellent measure of whether an application is "vertical" or "horizontal." Furthermore, it works independently of whether the program is sequential or parallel—it is just a measure of how CPU bound an application is, and so it does not matter whether the CPU is waiting for data from the memory system or from another node. Unfortunately, *calculating* β requires using application execution time for two frequencies, which requires at least two profile runs of the entire application. Below, we show two simple online metrics that *predict* the β of an application without such overhead.

6.1 Single-Node Programs

On a single-node program, there is no internode communication, and HPC applications are (generally) not I/O bound. Consequently, either the CPU or the memory system is on the critical path.

The first metric is MPO, which measures memory pressure.[1] This metric is determined using hardware performance counters to measure the number of operations retired and the L2 cache misses (which are memory accesses); both counters are initialized when the application commences and are read upon program termination. Therefore, MPO can be determined online. MPO stays constant as the frequency changes. The largest difference in MPO, using the same application and different frequencies, is about 1 percent. Therefore, MPO is more useful than the typical measure of IPC or UPC, which varies greatly with frequency. MPO correlates well to application slowdown (see below). It represents a good trade-off between 1) simplicity of data collection/analysis and 2) predictive power of the metric. As the MPO decreases, a program becomes more CPU bound because, on the average, it has fewer memory references per operation.

Fig. 10 shows a scatter plot of MPO versus β for all the applications we tested. In this and all figures that show MPO, for readability, we actually graph misses per *thousand* operations; this eliminates most decimal points. We ran a logarithmic regression to find the correlation between β, which is clearly a good metric (yet requires multiple runs), and $log(\mathrm{MPO})$. The figure plots the regression line, $0.697 - 0.150\log(\mathrm{MPO})$. We found that the correlation coefficient is 0.84 and the p-value[2] is 0.001, indicating a close correlation. Again, the advantage of MPO is that, unlike β, this measurement is obtained at runtime with negligible overhead using any gear.

TABLE 2
Comparing β to MPO and Slack (Sorted by β)

Benchmark	β	MPO	Slack	Benchmark	β	MPO	Slack
ft.A	0.052	1.92	0.877	sp.A	0.192	6.25	0.743
sp.B	0.072	6.18	0.604	bt.C	0.227	1.80	0.484
is.C	0.094	1.24	0.929	sp.C	0.231	6.17	0.541
bt.A	0.104	2.06	0.715	mg.B	0.466	2.17	0.228
is.A	0.128	1.24	0.930	bt.B	0.308	1.87	0.557
cg.A	0.141	1.64	0.830				

6.2 Multinode Programs

We now investigate multiple-node programs. Because the SPEC programs are sequential, we tested using NAS. However, we found that the correlation coefficient of MPO versus β for the multinode programs is just 0.52—this is not a close correlation. We conclude that using MPO is not sufficient to determine the proper gear setting on multinode programs. This is not surprising as, while β is a metric that implicitly includes all non-CPU activity (for example, memory accesses and communication latency), MPO only concerns memory pressure.

The additional metric we investigated to predict a bottleneck at runtime is *slack*, which predicts communication bottlenecks. The slack is simply the ratio of the total time a node is blocked in MPI calls to the total execution time. A larger slack suggests a greater benefit in reducing the gear. Similar to MPO, slack can be computed on the fly, with no additional program executions. In this paper, we measure slack by using our *MPI-Jack* tool, which intercepts MPI calls and allows for arbitrary code to be inserted before and after execution of the call. To determine blocking time, we simply take the wall clock time before and after MPI blocking calls, and blocking time is the difference between the measured times.

Hence, as described above, we need to include periods when the CPU is blocked and waiting for communicated data by using slack. We ran a multivariable correlation between β and *both* $\log(\mathrm{MPO})$ and slack. (Both $\log(\mathrm{MPO})$ and slack are independent variables, and β is the dependent variable.) Note that slack does not affect MPO; when a node is blocked, it does not do any operations at all. (The correlations show that multicollinearity is not a problem.)

Table 2 shows the values for β, MPO, and slack for all multinode programs that had relatively uniform slack between nodes (we did not use *lu* or the B and C classes of *cg*) running on eight nodes. The reason for requiring uniform slack is that reducing the gear in situations where some nodes have more slack than others does not yield a clean idea of what happens to β. This is discussed further below. Next, we added the single-node data points to the multinode set, setting slack to zero for the single-node points. This produced a correlation coefficient of 0.90,

1. The AMD executes a reduced-instruction-set computing (RISC) engine internally. It translates x86 (complex-instruction-set computing, CISC) instructions into RISC microoperations. Therefore, operations are a better indicator of CPU performance than instructions.

2. The p-value essentially gives statistical significance; it represents the probability that the relation between the variables is a mistake. Generally, the p-value should be less than 0.05.

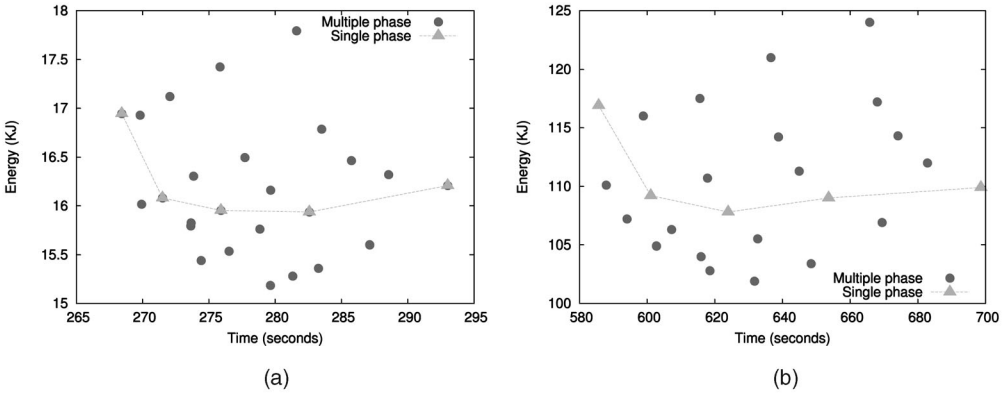

Fig. 11. Scatter plot of all possible gear permutations for SP class C (a) and a synthetic program (b). Circles indicate runs in which different gears were used for different phases and triangles indicate single-gear runs.

which is extremely good. Also, the p-values for $\log(\mathrm{MPO})$ and slack are both 0.001, which is excellent. We conclude that system software designers should be able to use metrics, such as MPO and slack, that can be collected on the fly to determine the proper energy gear for relatively load-balanced programs.

Finally, we discuss the situation when slack is not uniform between the nodes, as is the case when the load is not well balanced. The NAS programs CG and LU exhibit this property, along with Aztec and Sweep3d from the ASCI Purple suite. Here, the problem has several dimensions. First, different nodes need to be in different gears in the same code region, something that is outside the scope of this paper (which we addressed in [41]). Second, once the slack is balanced, then one needs to consider MPO. Again, this is outside our scope and, in fact, has not yet been addressed by the research community.

7 Discussion

This paper has focused on saving energy by executing programs at a lower energy gear. However, in the previous sections, we restricted our study in three ways. First, we used a single gear *per program*, as opposed to allowing the gear to vary dynamically. Second, we did not perform "field tests" that show how we would relate MPO to an actual gear selection based on a specific metric. Third, we did not discuss how to select the number of nodes to use. Although these have been the foci of our recent and current work, in this section, we discuss the basic ideas.

7.1 Phases

As MPO in large part determines the effectiveness of executing a program at a slower gear, different parts of programs, which often have different MPOs, may have different optimal gears. We refer to each distinct part of a program as a phase; many researchers have studied the general problem of phase detection (for example, [67], [42], [12], and [38]). In this work, we find phases as follows: We profile programs and divide them into phases based on changes in MPO. The profiling itself is done with *MPI-Jack*.

Fig. 11 shows two example programs, each with two phases, run in all possible gear permutations on a single node for the five fastest gears. The Fig. 11a is SP class C, where it is clear that phases are profitable. In particular, we can save almost 10 percent energy (which is double what is possible with a single gear solution) *and* run faster using phases than we can using a fixed gear solution. Fig. 11b is a synthetic program; the first phase is based on EP (small MPO), and the second phase is based on CG (large MPO). This represents a best case (when considering just a single node) for using phases. The results here show that we, again, can double the energy savings, this time up to at most 20 percent. Another way to look at this figure is that around 7 percent energy can be saved with essentially *no* time delay—if different gears are used (in this case, the first phase uses the fastest gear, and the second phase uses the second-fastest gear).

In Section 7.3, we show the use of dynamically switching gears in different phases in a multinode program. There, using phases in a situation where the total energy is limited can result in a much faster program.

7.2 Field Testing MPO

This section examines whether one can make practical use of our proposed metrics. As mentioned previously, frequency scaling is a mechanism that trades performance (time) for energy. One way to evaluate this trade-off is the *energy-delay product* (EDP) [33]. This product weighs energy savings and time equally. Thus, a decrease in energy is "canceled" by a proportional increase in time. To reduce the EDP, the relative energy decrease must be greater than the relative time increase. The previous section argued that MPO correlates to β, so it should be usable as a predictor of slowdown when using slower gears. Here, we use the data that we have collected to determine if MPO can be used to minimize EDP. In particular, we use the elapsed time and energy consumed for each program in each gear. From this, we can determine the gear that has the smallest EDP, which we call the "best" gear.

We selected several cutoff MPO values that partition the (energy-time) space into six distinct gear regions. If there exist cutoff points that also partition the space of the best gear, then MPO can be used to classify EDP. Fig. 12 shows this pictorially. It plots log(MPO) versus best gear for both SPEC sets (FP and INT) and NAS (sequential and parallel). The horizontal axis plots MPO—as in Section 6.1, it actually shows misses per *thousand* operations to eliminate most decimal points. The best gear is plotted on the vertical axis. Thus, each data point plots the MPO versus the gear that minimizes EDP. Notice that the best gear generally decreases as MPO increases. Furthermore, using EDP, the

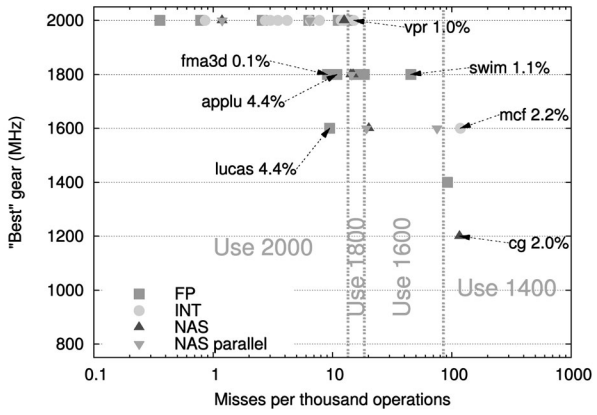

Fig. 12. Selecting EDP with MPO. Misclassified programs are labeled with the relative error.

best gear for most of the programs is the fastest gear (2,000 MHz).

We selected three cutoff points to divide the MPO space into four gears, which are 13.5, 18.5, and 85 (in MPO). Given this partitioning, we see that there are seven programs that do not execute in the best gear. These are enumerated in Fig. 12, along with the percentage error compared to the minimal EDP. Using this partitioning, MPO predicts the best gear 82 percent of the time. In only one case (*lucas*) does MPO fail to select a gear that is not adjacent to the best gear. In all cases where MPO does not predict the best gear, the error is less than 5 percent, and more than half the time, it is within 2 percent. For each program, the worst EDP is always a factor of 2 to 3 more than the best EDP; hence, although not perfect, this method clearly avoids poor gear choices.

EDP is only one way to evaluate the energy-time trade-off. Another choice is *energy-delay-squared product* (ED2P) [50]. This metric more heavily weighs the increase in time. In fact, the relative energy decrease has to more than double the relative time increase in order to reduce ED2P. Consequently, the top gear is best for 74 percent (28/38) of the programs. Fig. 13a shows the results of this partitioning. We found a partitioning that correctly classified 32 of the programs (84 percent), with a maximum error of 4.9 percent.

Using the above two metrics, the top gear is best for most of the programs, which may make it simple to predict the correct gear. For example, a trivial classification that always selects the top gear is correct 58 or 74 percent of the time for EDP or ED2P, respectively. Therefore, we also examine *energy-squared-delay product* (E2DP). In this case, the top gear is best for only 10 programs and 1,200 MHz is best for six programs. We found a partitioning in which 10 programs were misclassified (74 percent correct) and the maximum error was 7.2 percent, which is shown in Fig. 13b.

In summary, MPO can classify programs according to energy-time trade-off. However, the data are highly skewed to the top gear, so we do not make strong claims for its efficacy. Nevertheless, these data show that, while MPO *by itself* may not be an ideal classifier, it is an important metric that has a strong correlation to the best gear. We are currently investigating how to augment MPO to obtain a stronger classifier.

We note that we do not field test slack. This is because the choice of a gear on a given node for a given amount of slack is additionally dependent on the choice of gear on *other* nodes. Choosing proper gears in the presence of slack is the subject of the current work [41].

7.3 Number of Nodes

Traditionally, parallel computing users execute their programs on as many nodes as possible, each running as fast as possible. Sections 3 and 5 explain that programs may not run significantly faster when 1) running them at top speed per node or 2) running them on as many nodes as possible. The former has to do with low energy efficiency, whereas the latter is a traditional parallel computing issue—that of (low) parallel efficiency.

Fig. 14 shows the energy-time scatter plot for *lu* for all combinations of number of nodes and gear per phase. We denote the combination of a gear per phase and a number of nodes as a *schedule*. This benchmark has three phases. Each multigear schedule is shown by a triangle, whereas each single-gear schedule uses a square. The "dominating" points (ones that are better than all others in time for a given amount of energy) are connected by a dashed line. There are two things of note here. First, at each number of nodes, there are energy-efficient schedules that run almost as fast as running the schedule that uses the fastest gear throughout the program. In fact, if an energy limit was imposed, there exist certain limits (for example, 21.5 kJ), where using a multigear schedule leads to an execution time that is much less than that of the best single-gear schedule (about 37 to 57 seconds). Second, the parallel

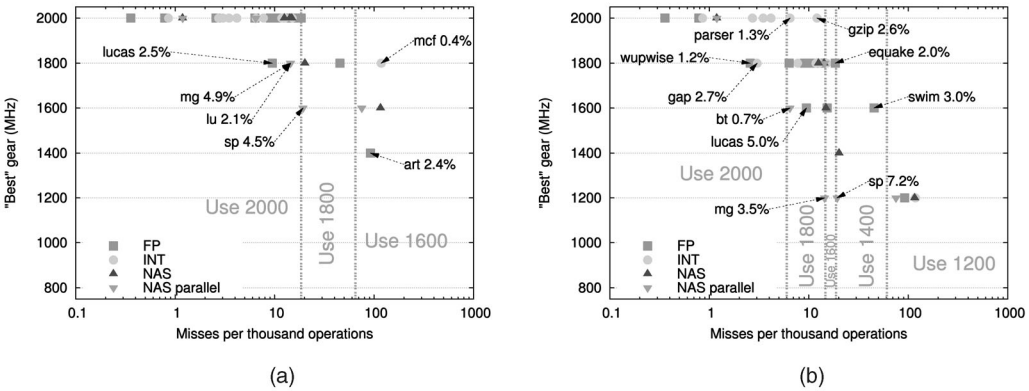

Fig. 13. Using MPO to determine the best gear using ED2P and E2DP. Misclassified programs are labeled with the relative error. (a) ED2P. (b) E2DP.

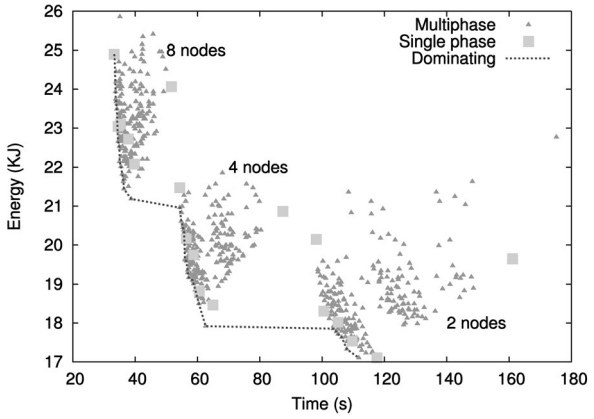

Fig. 14. Energy-time scatter plot of every LU schedule. For readability, the axes do not start at the origin.

efficiency is significantly less than 1 between four and eight nodes, because the time decreases by much less than a factor of two—so the energy consumed increases when comparing similar gear schedules.

Whether to use greater or fewer nodes as well as which gear per node requires understanding parallel and energy efficiency *simultaneously*. In general, this involves understanding the scalability of parallel programs, which is a well-studied yet difficult problem. By scalability, we mean both intranode (increasing the gear) and internode (increasing the number of nodes). We would like to be able to use an understanding of scalability to predict the graph shown in Fig. 14 so that an appropriate schedule (by some metric) can be chosen.

There are multiple ways that this can be done. One obvious way to understand scalability is to execute the program on every possible number of nodes in all possible gears. However, this is impractical—consider just the *lu* example and its 864 different schedules—so we have attacked this problem through modeling and partial program execution. Some researchers have studied scalability by investigating all message operations as the number of nodes increases [71].

Our current work involves using a combination of modeling, profiling, and partial program execution. The basic idea is to create a model of both execution time and consumed energy, make some partial program runs, and then use the model along with the profiling information to predict time and energy. This procedure is performed iteratively until a satisfactory schedule is found.

Full details on this can be found in [68]. Briefly, we mention the results that we found using this idea on *lu*. In just over 20 partial program executions under a specific energy limit[3] of 23 kJ, we found that the best schedule was to use all eight nodes, but to execute the three phases in gears 1,600, 1,200, and 1,600, respectively. This schedule was only 6 percent from optimal, which, again, would require 864 executions to find exhaustively.

3. This limit was set by taking the energy consumed at the fastest gear on eight nodes and decreasing it by 10 percent—which we did for all our tests [68].

8 Conclusions

This paper investigates the trade-off between energy and performance in both serial and parallel HPC applications. Using a wide range of applications, we found that, in the best case on one node, reducing the CPU speed makes it possible to use 20 percent less energy while increasing time by only 3 percent. On the other hand, a program that is largely CPU bound should be run at the fastest gear possible, because decreasing CPU speed may result in both slower execution and *more* energy. We examined two metrics, MPO and slack, which can be used by system software to choose the appropriate gear. We believe that choosing appropriate gears will be important in the future, where a cluster may have heat limitations.

In addition, we showed that avenues such as changing gears between phases and choosing the right number of nodes can further improve energy efficiency. Our current work involves both of these topics along with integrating solutions to them within MPI. This way, scientific programs already written using MPI will be more energy efficient.

References

[1] N.D. Adiga et al., "An Overview of the BlueGene/L Supercomputer," *Proc. Supercomputing '02,* Nov. 2002.

[2] ASCI Purple Benchmark Suite, http://www.llnl.gov/asci/platforms/purple/rfp/benchmarks/, year?

[3] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, "Non-Volatile Memory for Fast, Reliable File Systems," *Proc. Fifth Architectural Support for Programming Languages and Operating Systems Conf.,* 1992.

[4] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems," *Proc. Ninth ACM SIGOPS European Workshop,* Sept. 2000.

[5] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The Case of Power Management in Web Servers," *Power Aware Computing,* R. Graybill and R. Melham, eds., Kluwer/Plenum, 2002.

[6] E.V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," *Proc. Int'l Conf. Supercomputing,* pp. 86-97, 2003.

[7] J.S. Chase, D.C. Anderson, P.N. Thakar, A. Vahdat, and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proc. Symp. Operating Systems Principles,* pp. 103-116, 2001.

[8] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing Power with Performance Constraints for Parallel Sparse Applications," *Proc. First Workshop on High-Performance, Power-Aware Computing,* Apr. 2005.

[9] W. Chen, M. Dubios, and P. Stenstrom, "Integrating Complete-System and User-Level Performance/Power Simulators: The Simwattch Approach," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2003.

[10] Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, *Advanced Configuration and Power Interface Specification, Revision 2.0,* July 2000.

[11] V. Delaluz, A. Sivasubramanian, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, "Scheduler-Based DRAM Energy Management," *Proc. Design Automation Conf. (DAC '02),* June 2002.

[12] C. Ding and K. Kennedy, "Memory Bandwidth Bottleneck and Its Amelioration by a Compiler," *Proc. Int'l Parallel and Distributed Processing Symp.,* May 2000.

[13] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-Down Policies for Mobile Computers," *Proc. Second USENIX Symp. Mobile and Location-Independent Computing,* 1995.

[14] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers," *Proc. Fourth USENIX Symp. Internet Technologies and Systems (USITS '03),* 2003.

[15] E.N. (Mootaz) Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," *Proc. Workshop Mobile Computing Systems and Applications,* Feb. 2002.

[16] M.E. Femal, "Non-Uniform Power Distribution in Data Centers for Safely Overprovisioning Circuit Capacity and Boosting Throughput," master's thesis, North Carolina State Univ., Raleigh, N.C., May 2005.

[17] M.E. Femal and V.W. Freeh, "Safe Overprovisioning: Using Power Limits to Increase Aggregate Throughput," *Proc. Workshop Power-Aware Computer Systems,* Dec. 2004.

[18] M.E. Femal and V.W. Freeh, "Boosting Data Center Performance through Non-Uniform Power Allocation," *Proc. Second Int'l Conf. Autonomic Computing (ICAC),* June 2005.

[19] W.-C. Feng, "Making a Case for Efficient Supercomputing," *ACM Queue,* vol. 1, no. 7, Oct. 2003.

[20] X. Feng, R. Ge, and K.W. Cameron, "Power and Energy of Scientific Applications on Distributed Systems," *Proc. Int'l Parallel and Distributed Processing Symp.,* Apr. 2005.

[21] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic Performance-Setting for Dynamic Voltage Scaling," *Proc. MobiCom '01),* July 2001.

[22] V.W. Freeh, D.K. Lowenthal, R. Springer, F. Pan, and N. Kappiah, "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," *Proc. 19th Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* Apr. 2005.

[23] V.W. Freeh, D.K. Lowenthal, R. Springer, F. Pan, and N. Kappiah, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," *Principles and Practices of Parallel Processing,* June 2005.

[24] C. Gniady, Y.C. Hu, and Y.-H. Lu, "Program Counter Based Techniques for Dynamic Power Management," *Proc. 10th Int'l Symp. High-Performance Computer Architecture,* Feb. 2004.

[25] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *Proc. IEEE Int'l Symp. Low Power Electronics,* Oct. 1995.

[26] F. Gruian, "Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors," *Proc. Int'l Symp. Low-Power Electronics and Design (ISPLED '01),* Aug. 2001.

[27] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "Dynamic Speed Control for Power Management in Server Class Disks," *Proc. Int'l Symp. Computer Architecture,* pp. 169-179, June 2003.

[28] S. Gurumurthi, J. Zhang, A. Sivasubramaniam, M. Kandemir, H. Franke, N. Vijaykrishnan, and M.J. Irwin, "Interplay of Energy and Performance for Disk Arrays Running Transaction Processing Workloads," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* pp. 123-132, Mar. 2003.

[29] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "Reducing Disk Power Consumption in Servers with DRPM," *Computer,* pp. 41-48, Dec. 2003.

[30] T. Heath, B. Diniz, E.V. Carrera, W. Meira Jr., and R. Bianchini, "Energy Conservation in Heterogeneous Server Clusters," *Principles and Practice of Parallel Programming,* pp. 186-195, June 2005.

[31] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini, "Application Transformations for Energy and Performance-Aware Device Management," *Proc. 11th Int'l Conf. Parallel Architectures and Compilation Techniques,* Sept. 2002.

[32] D.P. Helmbold, D.D.E. Long, and B. Sherrod, "A Dynamic Disk Spin-Down Technique for Mobile Computing," *Mobile Computing and Networking,* pp. 130-142, 1996.

[33] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design," *Proc. Symp. Low Power Electronics,* pp. 8-11, Oct. 1994.

[34] C.-H. Hsu and U. Kremer, "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction," *Proc. ACM SIGPLAN Conf. Programming Languages, Design, and Implementation,* June 2003.

[35] C.-H. Hsu and W. Feng, "Effective Dynamic-Voltage Scaling through CPU-Boundedness Detection," *Proc. Fourth IEEE/ACM Workshop Power-Aware Computing Systems,* Dec. 2004.

[36] C.-H. Hsu and W. Feng, "Towards Efficient Supercomputing: Choosing the Right Efficiency Metric," *Proc. First Workshop High-Performance, Power-Aware Computing,* Apr. 2005.

[37] C. Hu, D. Jimenez, and U. Kremer, "Toward an Evaluation Infrastructure for Power and Energy Optimizations," *Proc. First Workshop High-Performance, Power-Aware Computing,* Apr. 2005.

[38] M. Huang, J. Renau, and J. Torellas, "Positional Adaptation of Processors: Application to Energy Reduction," *Proc. Int'l Symp. Computer Architecture,* pp. 157-168, June 2003.

[39] M. Huang, J. Renau, and J. Torrellas, "Profile-Based Energy Reduction in High-Performance Processors," *Proc. Fourth Workshop Feedback-Directed and Dynamic Optimizations,* Dec. 2001.

[40] R. Joseph and M. Martonosi, "Run-Time Power Estimation in High Performance Microprocessors," *Proc. Int'l Symp. Low-Power Electronics and Design (ISPLED '01),* Aug. 2001.

[41] N. Kappiah, V.W. Freeh, and D.K. Lowenthal, "Just in Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs," *Proc. Supercomputing,* Nov. 2005.

[42] K. Kennedy and U. Kremer, "Automatic Data Layout for Distributed-Memory Machines," *ACM Trans. Programming Languages and Systems,* vol. 20, no. 4, pp. 869-916, 1998.

[43] R. Kotla, S. Ghiasi, T. Keller, and F. Rawson, "Scheduling Processor Voltage and Frequency in Server and Cluster Systems," *Proc. Workshop High-Performance, Power-Aware Computing,* 2005.

[44] R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access with Bounded Slowdown," *Proc. MobiCom '02,* Sept. 2002.

[45] R. Kravets, K. Schwan, and K. Calvert, "Power-Aware Communication for Mobile Computers," *Proc. Sixth Int'l Workshop Mobile Multimedia Comm.,* Nov. 1999.

[46] A.R. Lebeck, X. Fan, H. Zeng, and C.S. Ellis, "Power Aware Page Allocation," *Architectural Support for Programming Languages and Operating Systems,* pp. 105-116, 2000.

[47] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller, "Energy Management for Commercial Servers," *Computer,* pp. 39-48, Dec. 2003.

[48] D. Li and J. Wang, "Eeraid: Energy Efficient Redundant and Inexpensive Disk Array," *Proc. SIGOPS European Workshop,* Sept. 2004.

[49] K. Li, R. Kumpf, P. Horton, and T.E. Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers," *Proc. USENIX Winter,* pp. 279-291, 1994.

[50] A.J. Martin, M. Nystroem, and P. Penzes, "ET2: A Metric for Time and Energy Efficiency of Computation," Technical Report CSTR:2001.007, California Inst. Technology, Pasadena, Calif., 2001.

[51] M. Martonosi, D. Brooks, and P. Bose, "Modeling and Analyzing CPU Power and Performance: Metrics, Methods, and Abstractions," *Proc. SIGMETRICS,* 2001.

[52] WattsUp Meters, http://www.doubleed.com, 2007.

[53] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling," *Proc. 16th Int'l Conf. Supercomputing,* pp. 35-44, 2002.

[54] Orion Multisystems, http://www.orionmulti.com/, 2006.

[55] S. Palacharla, N.P. Jouppi, and J.E. Smith, "Complexity-Effective Superscalar Processors," *Proc. 24th Ann. Int'l Symp. Computer Architecture (ISCA '97),* pp. 206-218, 1997.

[56] F. Pan, "Exploring the Energy-Time Tradeoff in High Performance Computing," master's thesis, North Carolina State Univ., Raleigh, N.C., May 2005.

[57] F. Pan, V.W. Freeh, and D.M. Smith, "Exploring the Energy-Time Tradeoff in High Performance Computing," *Proc. Workshop High-Performance, Power-Aware Computing,* Apr. 2005, in conjunction with the *Proc. Int'l Parallel and Distributed Processing Symp.*

[58] A.E. Papathanasiou and M.L. Scott, "Energy Efficiency through Burstiness," *Proc. IEEE Workshop Mobile Computing Systems and Applications,* Oct. 2003.

[59] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *Proc. Int'l Symp. Low-Power Electronics and Design (ISPLED '98),* pp. 76-81, Aug. 1998.

[60] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath, "Dynamic Cluster Reconfiguration for Power and Performance," *Compilers and Operating Systems for Low Power,* Sept. 2001.

[61] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-Based Servers," *Proc. 18th Ann. Int'l Conf. Supercomputing,* pp. 68-78, June 2004.

[62] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," *Proc. Workshop Compilers and Operating Systems for Low Power,* Sept. 2001.

[63] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath, "Dynamic Cluster Reconfiguration for Power and Performance," *Compilers and Operating Systems for Low Power,* pp. 75-93, Kluwer Academic Publishers, 2003.

[64] J. Pouwelse, K. Langendoen, and H. Sips, "Energy Priority Scheduling for Variable Voltage Processors," *Proc. Int'l Symp. Low-Power Electronics and Design (ISPLED '01),* Aug. 2001.
[65] M. Schulz, B. White, S. McKee, H. Lee, and J. Jeitner, "Owl: Next-Generation System Monitoring," *ACM Computing Frontiers,* pp. 116-124, May 2005.
[66] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron, "Power-Aware QoS Management in Web Servers," *Proc. 24th Ann. IEEE Real-Time Systems Symp.,* Dec. 2003.
[67] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* Oct. 2002.
[68] R.C. Springer IV, D.K. Lowenthal, B. Rountree, and V.W. Freeh, "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming,* Mar. 2006.
[69] A. Vahdat, A. Lebeck, and C. Ellis, "Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency," *Proc. SIGOPS European Workshop,* 2000.
[70] M. Valluri and L.K. John, "Is Compiling for Performance = Compiling for Power?" *Proc. Fifth Ann. Workshop Interaction between Compilers and Computer Architectures,* 2001.
[71] J.S. Vetter, "Performance Analysis of Distributed Applications Using Automatic Classification of Communication Inefficiencies," *Proc. Int'l Conf. Supercomputing,* pp. 245-254, May 2000.
[72] M. Warren, E. Weigle, and W. Feng, "High-Density Computing: A 240-Node Beowulf in One Cubic Meter," *Proc. Supercomputing '02,* Nov. 2002.
[73] A. Weissel, B. Beutel, and F. Bellosa, "Cooperative IO—A Novel IO Semantics for Energy-Aware Applications," *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI '02),* Dec. 2002.
[74] J. Wilkes, "Predictive Power Consumption," Technical Report HPL-CSP-92-5, Hewlett-Packard Labs, Feb. 1992.
[75] F.C. Wong et al., "Architectural Requirements and Scalability of the NAS Parallel Benchmarks," *Proc. Supercomputing '99,* Nov. 1999.
[76] H. Yan, R. Krishnan, S.A. Watterson, D.K. Lowenthal, K. Li, and L.L. Peterson, "Client-Centered Energy and Delay Analysis for TCP Downloads," *Proc. 14th IEEE Int'l Workshop Quality of Service,* June 2004.
[77] H. Zeng, C.S. Ellis, A.R. Lebeck, and A. Vahdat, "Currentcy: Unifying Policies for Resource Management," *Proc. USENIX 2003 Ann. Technical Conf.,* June 2003.
[78] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wikes, "Hibernator: Helping Disk Arrays Sleep through the Winter," *Proc. Symp. Operating Systems Principles,* pp. 177-190, Oct. 2005.
[79] Q. Zhu, F.M. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," *Proc. 10th Int'l Symp. High-Performance Computer Architecture (HPCA-10),* Feb. 2004.

Vincent W. Freeh received the PhD degree from the University of Arizona in 1996. He is an assistant professor of computer science at North Carolina State University (NCSU). He was a captain in the US Army Corps of Engineers before entering graduate school for the MS degree. He worked at IBM in the Storage System Division until he returned to school to earn his PhD degree. He was on the faculty of the University of Notre Dame prior to coming to NCSU. His research interest is high-performance system software with an emphasis on file systems, parallel and distributed systems, and power-aware computing. He received a US National Science Foundation CAREER Award and an IBM Faculty Development Award.

David K. Lowenthal received the PhD degree from the University of Arizona in 1996. He is an associate professor of computer science at the University of Georgia. His research centers on parallel and distributed computing, operating systems, and networks. His current research projects include addressing scalability and energy for high-performance computing as well as developing an infrastructure for flexible TCP-compatible protocols.

Feng Pan received the BS degree in computer science from the Appalachian State University in 2003. He went to graduate school at North Carolina State University. He is currently self-employed.

Nandini Kappiah received the BEng degree in information science from Visweswaraiah Technological University, Bangalore, in 2003 and the MS degree in computer science from North Carolina State University in 2005. Her master's thesis, "Just-in-Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs," was in the field of high-performance power-aware computing. She has been a software engineer at Google Inc. since August 2005.

Rob Springer received the MS degree in computer science from the University of Georgia in 2005, under the direction of David K. Lowenthal. He is currently a software engineer at Philips Medical Systems. Aside from parallel systems, his work and research interests include embedded development, operating systems, and game engine architecture.

Barry L. Rountree received the BA degree in theater from the Ohio University Honors Tutorial College and the MS degree in network and system administration from Florida State University. He is currently working toward the PhD degree at the University of Georgia, under the direction of Dr. David K. Lowenthal.

Mark E. Femal received the BS degree in computer science from the University of Wisconsin-Oshkosh in 1998. After working in the telecommunications and insurance industries for five years, he subsequently completed the MS degree in computer science from North Carolina State University in 2005. He is currently with the US Navy as a computer scientist at the Stennis Space Center, Mississippi. He is a member of the IEEE and the ACM.