

# Analyzing the Energy-Time Tradeoff in High-Performance Computing Applications

Vincent W. Freeh<sup>†</sup> Feng Pan<sup>†</sup> David K. Lowenthal<sup>‡</sup> Nandini Kappiah<sup>†</sup>  
Rob Springer<sup>‡</sup> Barry L. Rountree<sup>‡</sup> Mark E. Femal<sup>†</sup>

<sup>†</sup>Department of Computer Science  
North Carolina State University, Raleigh, NC  
{vwfreeh,fpan2,nkappia,mefemal}@ncsu.edu

<sup>‡</sup>Department of Computer Science  
The University of Georgia, Athens, GA  
{dkl,rountree,springer}@cs.uga.edu

## 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 tradeoff can be dynamically adjusted.

This paper analyzes the energy-time tradeoff 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 run time, 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.

## 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: *i.e.*, the last 10% increase in performance requires a disproportionately large amount of resources. The Earth Simulator, one of the world’s fastest supercomputers, consumes 7 megawatts 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 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 tradeoff between energy and performance (execution time) for both serial and parallel HPC programs. Our results illustrate an *energy-time tradeoff*: by this, we mean that decreasing energy is possible at the cost of increased time. Not every energy-time tradeoff 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 (*e.g.*, 10% less energy and 10% more time), and some are much better. For example, we found that the single-node version of *cg* from the NAS suite allowed 20% less energy to be used while only increasing execution time by 3%. 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* and *slack*, that can predict this energy-time tradeoff. We argue that these metrics will allow system software designers to determine when to use frequency scaling and which gear to use.

The rest of this paper is organized as follows. The next section describes related work, and Section 3 describes our methodology. We show results for a single and multiple nodes, respectively, in Sections 4 and 5. Next, Section 6 discusses categorizing applications for the purpose of choosing the proper energy gear using on-line 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 policies, was investigated in [15].

Such work shows that power and energy management are 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 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 speedup an application, which is often highly regular and predictable. One approach is to save energy in an application-specific way; [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, [48] looked at new scheduling and caching schemes, and [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 data is migrated to a disk appropriate for that data. 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 [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/run-time approach using this was designed and implemented in [31], and a prefetching approach in [58].

There are also a few high-performance computing clusters designed with energy in mind. One is Blue-Gene/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, 55]. In addition, IPC is utilized in [39] for an architectural simulator for both dynamic energy efficiency and temperature management. Kotla *et al.* uses IPC as a means of predicting future performance to schedule frequency changes in a server cluster [43]. Section 6 discusses an alternative metric (misses per operations) 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 tradeoff in mobile systems. In [22] we study the energy-time tradeoff in 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 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. This subsection details some of these projects.

ECOSystem [77] attempts to implement a power management system without the need to rewrite appli-

cation software using the *currentcy* 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 ACPI (Advanced Configuration and Power Interface) standard [10]. It is an evolution of several existing methods including BIOS power management, the APM (Advanced Power Management) API, and a smart battery interface.

In general, the goal of the OS is to reduce energy given a multiprogrammed workload; while 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.

**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 dynamic voltage scaling (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.

**Disk** Disks consume a large percentage of energy on some mobile architectures. Many have studied disk spindown to save energy (*e.g.*, [32, 13, 74, 3, 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.

**Memory and Network** In some architectures, individual memory banks and network cards have multiple energy states. Work on transitioning these to lower energy states includes [11, 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, 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, SPEC integer, and SPEC floating point. The NAS suite is a popular high-performance computing benchmark. It consists of scientific benchmarks including application areas such as sorting, spectral transforms, and fluid dynamics. In contrast, the SPEC integer benchmarks are non-scientific applications that are CPU and/or memory intensive. The SPEC floating point benchmarks are a mixture of both scientific and non-scientific programs. For example, *mesa* and *facerec* are non-scientific, graphics programs, whereas *swim* and *mgrid* are well-known scientific benchmarks. In our multi-node tests, we studied the NAS MPI suite as well as some of the ASCI benchmarks [2]. We used all valid configurations

<i>Frequency</i> (MHz)	<i>Voltage</i> (V)	<i>Power (W)</i>	
		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

Table 1: Idle and active (total system) power for each AMD-64 node.

up to 9 nodes. Presumably, such mature benchmarks have been thoroughly analyzed and are well-written (e.g., 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 ten 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 MHz to 2000 MHz in steps of 200 MHz. The voltage, which ranges from 1.5–1.0V, is reduced in each gear. Each node has 1GB main memory, a 128KB L1 cache (split), and a 512KB L2 cache. The nodes are connected by 100Mb/s 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 execution time and energy consumed for an application at a range of energy gears. Execution time is elapsed wall clock time. The energy consumed by the entire system is calculated by measuring 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 multi-node 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 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%, while the difference in idle power is approximately 15%.

The time to shift between any two gears is at most 500 microseconds and generally less than half that [16]. Figure 1 shows that overhead of gear shifting is acceptably small, for the range of shifting frequencies shown. The energy consumption and elapsed time of the *lu* benchmark from the NAS suite is shown for a single node in the cluster. The plots show these values for various shifting frequencies. For example, the left-most 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 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% of the average value (note that the extreme lines differ by 5% and 8% in time and energy, respectively).

The next section shows the single-node results, which illustrate the energy-time tradeoff 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 tradeoff when the number of nodes increases.

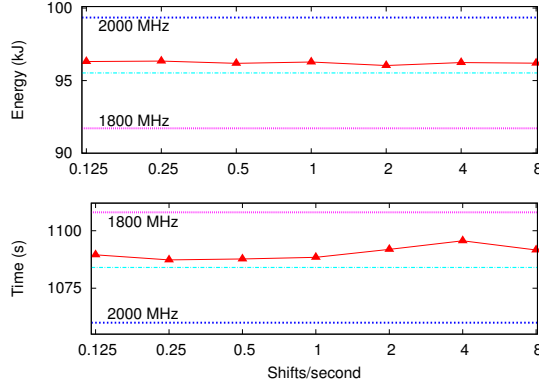


Figure 1: Examining the effects of shifting gears on *lu*.

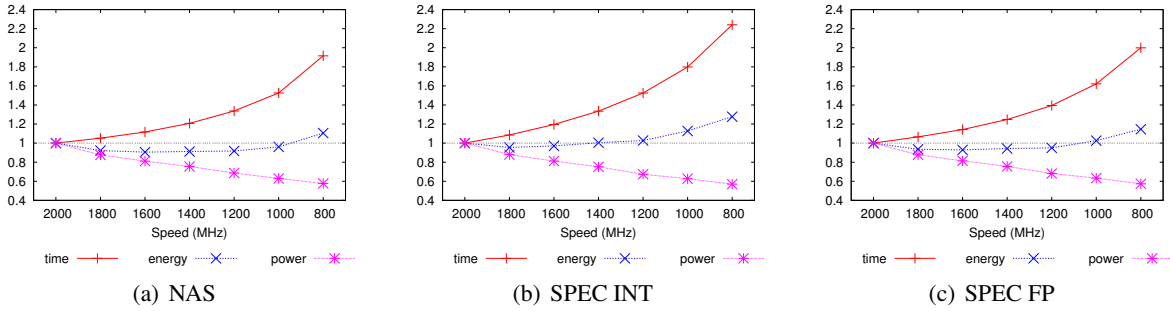


Figure 2: Aggregate plots of all programs in each set.

## 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].

### 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.

Figure 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 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, *etc.* 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 tradeoff: there is an energy savings (and a time delay)

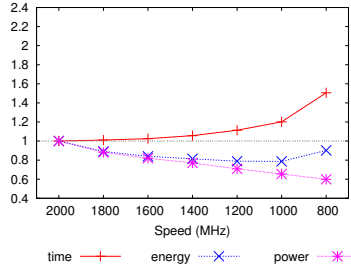


Figure 3: Aggregate plot of the three NAS programs that have least CPU bound.

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 1600 MHz, the energy used and time taken are 91% and 112% 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 below. While the SPEC sets are common benchmark suites, many of the programs are not representative of high-performance computing 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, Figure 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% of full, respectively, at 1600 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 tradeoff 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 Figure 4 have the best energy-time tradeoff 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 Figure 4(a). Setting the gear to 1400 MHz yields a 3% increase in execution time compared to the fastest, while the corresponding decrease in energy consumption is 20%.

Next, we examine how a vertical energy-time shape occurs. Our experience shows that programs use essentially the same number (within 1%) of micro-operations 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 (2000 and 1800 MHz), in which the performance gain is less than 1%. Using the slower gear with a clock rate that is 90% of the highest, the execution has 90% as many cycles (approximately 5.0 to 4.5 trillion). Because the number of micro-operations does not change, the microprocessor throughput, in micro-operations 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,

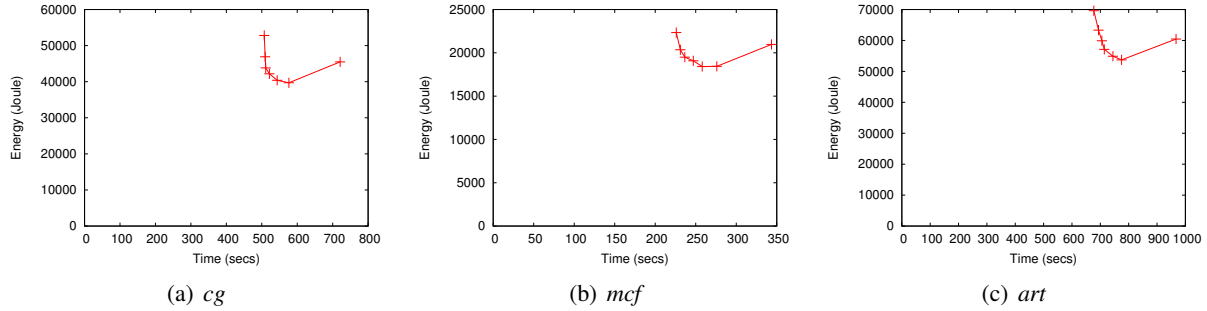


Figure 4: Best energy-time tradeoff in each set.

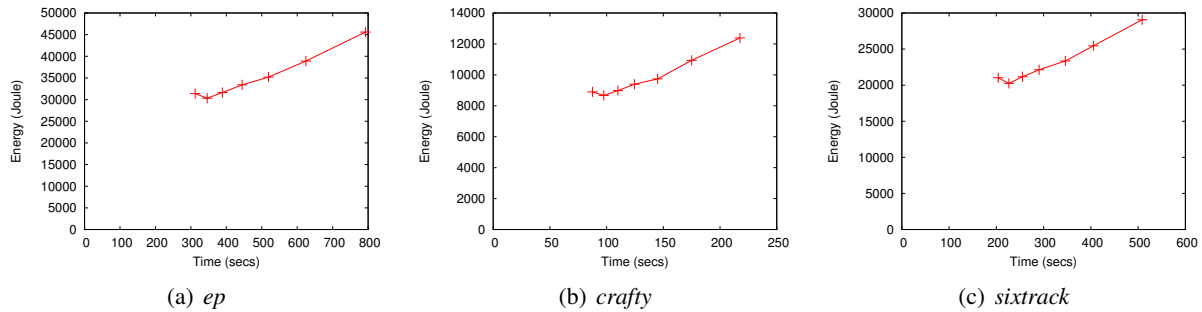


Figure 5: Worst energy-time tradeoff in each set.

determine that memory is the bottleneck.

On the other hand, Figure 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% for all three programs, which is nearly the same as the increase in the CPU cycle time (11%). Thus, these programs are highly CPU bound. The minimum energy is always at 1800 MHz, and the savings is less than 4%. The reason the energy decreases is that the power consumption decreases by 13 to 14%, which is more than the time increases. Therefore, the last 10% of performance requires more than 10% more power.

## 5 Multiple-Node Results

The previous section investigated the energy-time tradeoff on a single node. This section studies the effect of distributed programs. Figure 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 Figure 4 and Figure 5, except that it shows the results from multiple experiments: 2, 4, and 8 nodes (or 4 and 9 nodes in the case of *bt* and *sp*). It also plots the one-node results from the previous section, 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

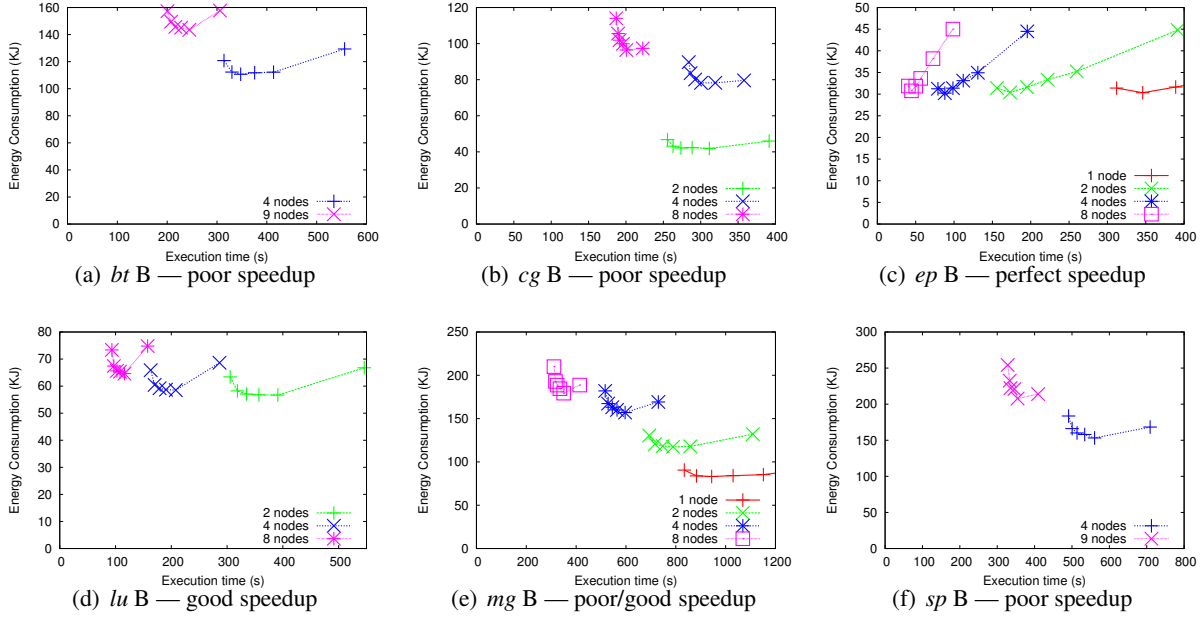


Figure 6: Energy consumption vs. execution time for NAS benchmarks for 1 to 8 or 9 nodes.

in a distributed program, not only might a processor wait for the memory subsystem, but at times it also might 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, 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 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 not an energy-time tradeoff between these points because one point dominates the other in both energy and time. This case occurs when **speedup is good** (*i.e.*, 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.

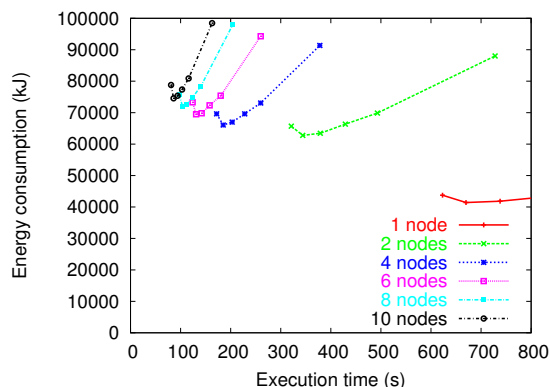


Figure 7: Energy consumption vs. execution time for Jacobi iteration on 2, 4, 6, 8, and 10 nodes.

### Case 1: Poor Speedup

Figure 6 offers several examples of case 1. In particular, this case is illustrated in *bt*, *sp*, and *mg* from 2 to 4 nodes, and *cg* both from 2 to 4 and 4 to 8 nodes (the 2 to 4 case for *cg* actually slowed down). Suppose a given supercomputer cluster were 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

Figure 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 tradeoff.

### Case 3: Good Speedup

Figure 6 shows several examples of this case. First, consider *lu* at 4 and 8 nodes. Gear 4 on 8 nodes uses approximately the same energy as the fastest gear on 4 nodes, but executes 50% more quickly. The fastest gear executes 72% faster on 8 nodes than on 4 nodes, but uses 12% more energy. This case illustrates an additional choice not available in a conventional cluster, which only supports either the fastest gear option (4 or 8 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 8 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, Figure 7 plots data for 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 5 configurations: 2, 4, 6, 8, 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 in case 3. For example, executing in the second or third fastest gear on 6 nodes results in the program finishing faster *and* using less energy than using fastest gear on 4 nodes.

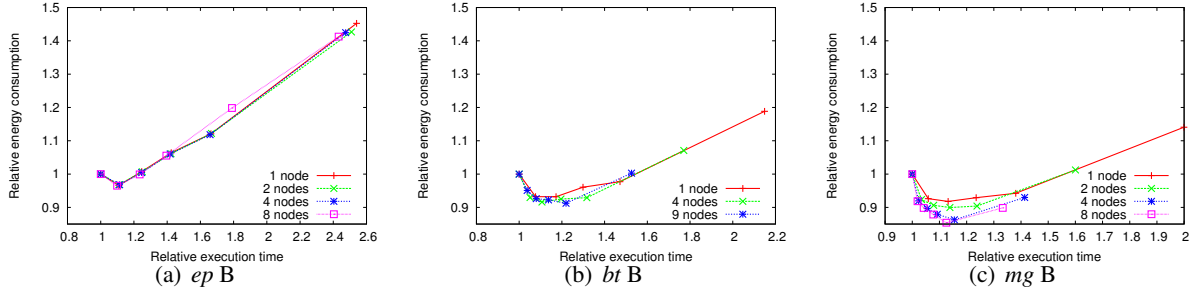


Figure 8: Relative energy vs. time.

## 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. Figure 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. Figure 8(a) shows that for *ep*, increasing the number of nodes has little effect on the *relative* energy or time. This is because *ep* has almost perfect speedup. Consequently, the program executes nearly identically on any number of nodes.

The next case is illustrated in Figure 8(b). This program is *bt*, which speeds up poorly because it has a large communication component. While an increase in CPU cycle time increases computation time, it has little effect on the communication time. Therefore, the increase in execution time due to frequency reduction is less significant as the computation to communication ratio decreases. This effect is shown in Figure 8(b), 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 (Figure 8(c)). 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% on 4 and 9 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% on 2, 4, and 8 nodes, respectively); therefore, the gear setting makes a more pronounced difference.

## 5.3 Additional Applications

Figure 9 shows the energy-time tradeoff 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

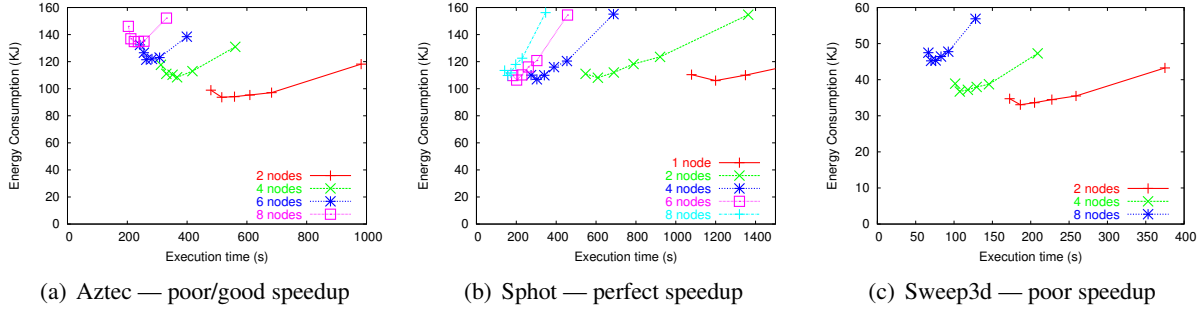


Figure 9: Energy-time tradeoff of some applications.

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, while programs like *ep* should be run at fastest gear on as many nodes as possible.

The question is: which metrics should be used to categorize programs into equivalence classes? As explained in the last two sections, 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 (e.g., [65]).

First, we investigated a metric termed  $\beta$ , introduced by Hsu and Kremer in [34] and applied to HPC by Hsu and Feng [35], that 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 (2000MHz in this case). The criticality of the CPU,  $\beta$ , 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  $\beta$  values (same application, but difference frequency) is 5%.

The NAS suite has an average  $\beta$  of 0.40, and half of those programs are less than 0.50. On the other hand, the SPEC suites have a  $\beta$  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  $\beta$  less than 0.50. This indicates that the NAS programs are less CPU bound than SPEC.

In general,  $\beta$  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

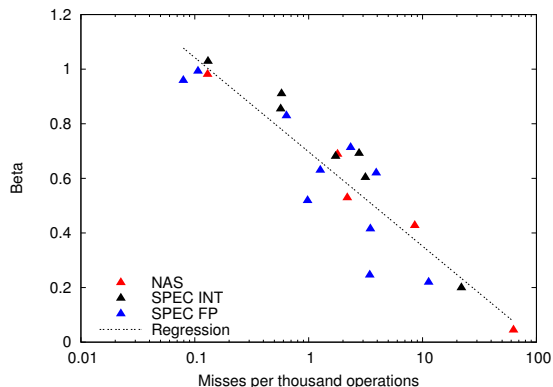


Figure 10: MPO vs.  $\beta$  for all single-node applications on AMD-64.

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*  $\beta$  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  $\beta$  of an application without such overhead.

## 6.1 Single-node programs.

On a single-node program, there is no inter-node 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 *misses per operation* (MPO), which measures memory pressure.<sup>1</sup> 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 on line. MPO stays constant as the frequency changes. The largest difference in MPO, using the same application and different frequencies, is about 1%. Therefore, MPO is more useful than the typical measure of IPC or UPC, which vary greatly with frequency. MPO correlates well to application slowdown (see below). It represents a good tradeoff 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 average, it has fewer memory references per operation.

Figure 10 shows a scatter plot of MPO vs.  $\beta$  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  $\beta$ , which is clearly a good metric (yet requires multiple runs), and  $\log(\text{MPO})$ . The figure plots the regression line,  $0.697 - 0.150 \log(\text{MPO})$ . We found that the correlation coefficient is 0.84 and the p-value<sup>2</sup> is 0.001, indicating a close correlation. Again, the advantage of MPO is that unlike  $\beta$ , this measurement is obtained at runtime with negligible overhead using any gear.

<sup>1</sup>The AMD executes a RISC engine internally. It translates x86 (CISC) instructions into RISC micro-operations. Therefore, operations are a better indicator of CPU performance than instructions.

<sup>2</sup>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.

Benchmark	$\beta$	MPO	Slack	Benchmark	$\beta$	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				

Table 2: Comparing  $\beta$  to MPO and slack (sorted by  $\beta$ ).

## 6.2 Multi-node 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 vs.  $\beta$  for the multi-node 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 multi-node programs. This is not surprising, as while  $\beta$  is a metric that implicitly includes all non-CPU activity (e.g., memory accesses, 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 wallclock 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 multi-variable correlation between  $\beta$  and *both*  $\log(\text{MPO})$  and slack. (Both  $\log(\text{MPO})$  and slack are independent variables, and  $\beta$  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  $\beta$ , MPO, and slack for all multi-node 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  $\beta$ . This is discussed further below. Next, we added the single-node data points to the multi-node set, setting slack to zero for the single-node points. This produced an correlation coefficient of 0.90, which is extremely good; Also, the p-values for  $\log(\text{MPO})$  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 (that 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.

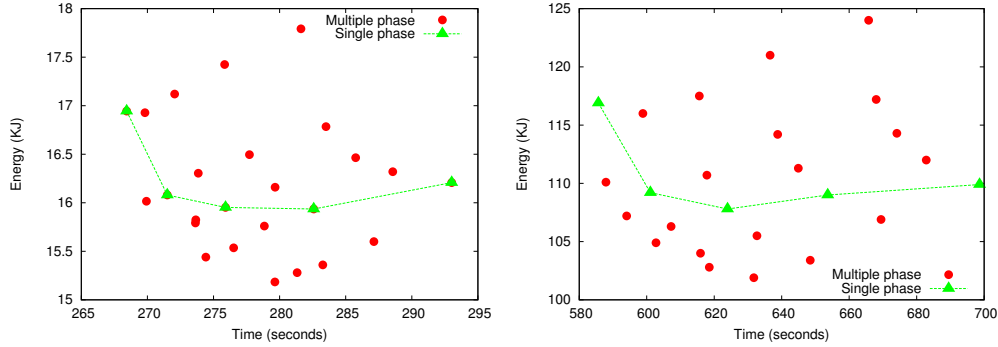


Figure 11: Scatter plot of all possible gear permutations for SP class C (left) and a synthetic program (right). Circles indicate runs in which different gears were used for different phases, and triangles indicate single-gear runs.

## 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. While 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 (e.g., [67, 42, 12, 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*.

Figure 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 left-hand figure is SP class C, where it is clear that phases are profitable. In particular, we can save almost 10% 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. The right-hand figure 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 of 20%. Another way to look at this figure is that around 7% 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 multi-node program. There, using phases in a situation where 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 tradeoff is the *energy-delay product* (EDP) [33]. This product weighs energy savings and time equally. Thus

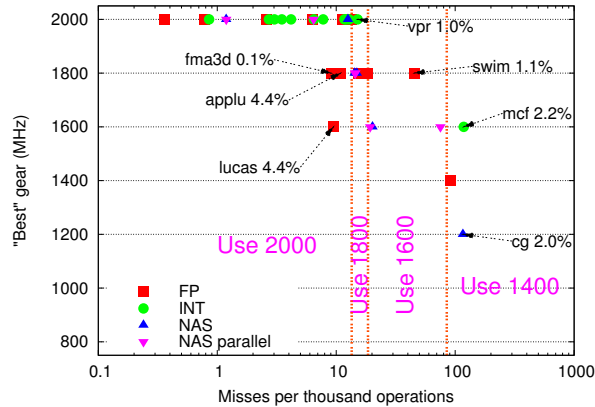


Figure 12: Selecting EDP with MPO. Misclassified programs are labeled with the relative error.

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  $\beta$ , so it should be usable as a predictor of slowdown when using slower gears. Here, we use the data 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 6 distinct gear regions. If there exist cutoff points that also partition the space of best gear, then MPO can be used to classify EDP. Figure 12 shows this pictorially. It plots  $\log(\text{MPO})$  vs. best gear for both SPEC sets (FP and INT) and NAS (sequential and parallel). The horizontal axis plots MPO—as in the previous section, 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 best gear for most of the programs is the fastest gear (2000 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 7 programs that do not execute in the best gear. These are enumerated in Figure 12 along with the percentage error compared to the minimal EDP. Using this partitioning, MPO predicts the best gear 82% of 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%, and more than half the time it is within 2%. For each program the worst EDP is always a factor of 2 to 3 more than the best EDP, so while not perfect, this method clearly avoids poor gear choices.

EDP is only one way to evaluate the energy-time tradeoff. 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, top gear is best for 74% (28/38) of the programs. Figure 13(a) shows the results of this partitioning. We found a partitioning that correctly classified 32 of the programs (84%) with maximum error of 4.9%.

Using the above two metrics, 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 top gear is correct 58% or 74% of the time for EDP or ED2P, respectively. Therefore, we also examine *energy-squared-delay product* (E2DP). In this case, top gear is best for only 10 programs and 1200MHz is best for 6 programs. We found a partitioning in which 10 programs were misclassified (74% correct) and the maximum error was 7.2%, which is shown in Figure 13(b).

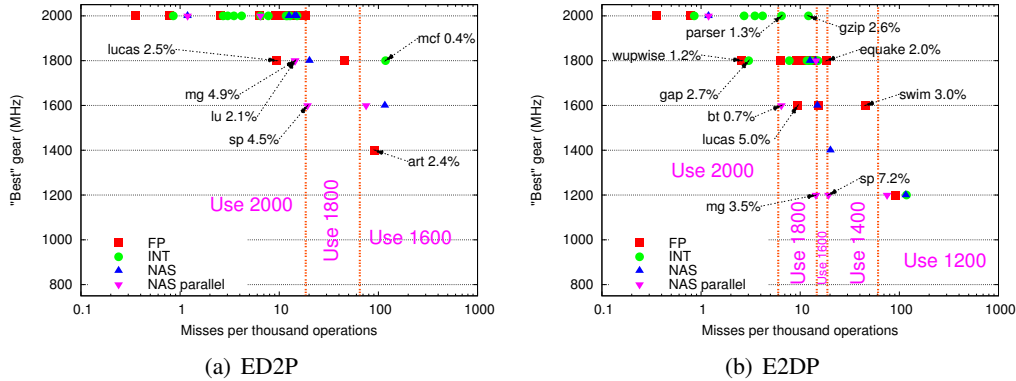


Figure 13: Using MPO to determine the best gear using ED2P and E2DP. Misclassified programs are labeled with the relative error.

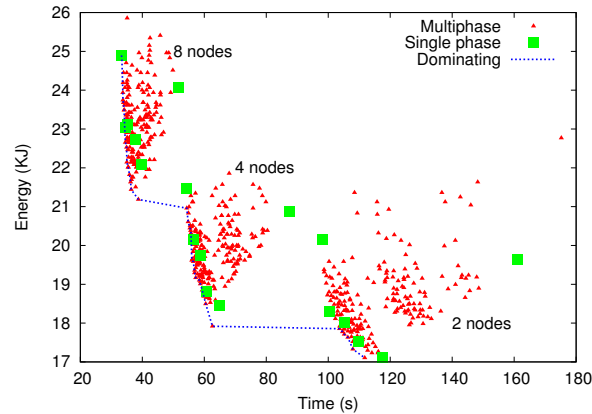


Figure 14: Energy-time scatter plot of every LU schedule. For readability, the axes do not start at the origin.

In summary, MPO can classify programs according to energy-time tradeoff. However, the data is highly skewed to the top gear, so we do not make strong claims for its efficacy. Nevertheless, this data show that while MPO *by itself* may not be an ideal classifier, it also shows that it is an important metric that has a strong correlation to 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 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, while the latter is a traditional parallel computing issue—that of (low) parallel efficiency.

Figure 14 shows the energy-time scatterplot 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 multi-gear schedule is shown by a triangle, while each single-gear schedule uses a

square. The “dominating” points (ones that are better than all others in time for 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 were imposed, there exist certain limits (for example, 21.5 KJ) where using a multi-gear schedule leads to an execution time that is much less than that of the best single-gear schedule (about 37 seconds to 57 seconds). Second, the parallel efficiency is significantly less than 1 between 4 and 8 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 scalability of parallel programs, which is a well-studied yet difficult problem. By scalability we mean both intra-node (increasing the gear) and inter-node (increasing the number of nodes). We would like to be able to use an understanding of scalability to predict the graph shown in Figure 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 we found using this idea on *lu*. In just over 20 partial program executions under a specific energy limit<sup>3</sup> of 23 KJ, we found that the best schedule was to use all 8 nodes, but to execute the three phases in gears 1600, 1200, and 1600, respectively. This schedule was only 6% from optimal, which again would require 864 executions to find exhaustively.

## 8 Conclusions

This paper investigates the tradeoff 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% less energy while increasing time by only 3%. 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, misses per operation 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.

---

<sup>3</sup>This limit was set by taking the energy consumed at fastest gear on 8 nodes and decreasing it by 10%—which we did for all our tests [68].

## References

- [1] N.D. Adiga et al. An overview of the BlueGene/L supercomputer. In *Supercomputing 2002*, November 2002.
- [2] ASCI Purple Benchmark Suite. <http://www.llnl.gov/asci/platforms/purple/rfp/benchmarks/>.
- [3] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of the 5th ASPLOS*, 1992.
- [4] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000.
- [5] Pat Bohrer, Elmootazbellah Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The case of power management in web servers. In Robert Graybill and Rami Melham, editors, *Power Aware Computing*. Kluwer/Plenum, 2002.
- [6] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *Proceedings of International Conference on Supercomputing*, pages 86–97, San Fransisco, CA, 2003.
- [7] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.
- [8] Guilin Chen, Konrad Malkowski, Mahmut Kandemir, and Padma Raghavan. Reducing power with performance constraints for parallel sparse applications. In *First Workshop on High-Performance, Power-Aware Computing*, April 2005.
- [9] W. Chen, M. Dubios, and P. Stenstrom. Integrating complete-system and user-level performance/power simulators: The simwatch approach. In *International Symposium on 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. In *Proc. Design Automation Conf. (DAC '02)*, Jun 2002.
- [12] Chen Ding and Ken Kennedy. Memory bandwidth bottleneck and its amelioration by a compiler. In *International Parallel and Distributed Processing Symposium*, May 2000.
- [13] F. Douglass, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.
- [14] Elmootazbellah Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *USITS '03*, 2003.
- [15] E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Workshop on Mobile Computing Systems and Applications*, Feb 2002.

- [16] Mark E. Femal. Non-uniform power distribution in data centers for safely overprovisioning circuit capacity and boosting throughput. Master's thesis, North Carolina State University, Raleigh, NC, May 2005.
- [17] Mark E. Femal and Vincent W. Freeh. Safe overprovisioning: Using power limits to increase aggregate throughput. In *Workshop on Power-Aware Computer Systems*, Portland, OR, December 2004.
- [18] Mark E. Femal and Vincent W. Freeh. Boosting data center performance through non-uniform power allocation. In *Second International Conference on Autonomic Computing (ICAC)*, Seattle, WA, June 2005.
- [19] Wu-Chun Feng. Making a case for efficient supercomputing. *ACM Queue*, 1(7), October 2003.
- [20] X. Feng, R. Ge, and K. W. Cameron. Power and energy of scientific applications on distributed systems. In *International Parallel and Distributed Processing Symposium*, April 2005.
- [21] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the 7th Conference on Mobile Computing and Networking MOBICOM '01*, July 2001.
- [22] Vincent W. Freeh, David K. Lowenthal, Rob Springer, Feng Pan, and Nandani Kappiah. Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In *IPDPS 2005*, Denver, CO, April 2005.
- [23] Vincent W. Freeh, David K. Lowenthal, Rob Springer, Feng Pan, and Nandani Kappiah. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Principles and Practices of Parallel Processing*, June 2005.
- [24] Chris Gniady, Y. Charlie Hu, and Yung-Hsiang Lu. Program counter based techniques for dynamic power management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture*, February 2004.
- [25] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. In *IEEE International Symposium on Low Power Electronics*, October 1995.
- [26] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.
- [27] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Dynamic speed control for power management in server class disks. In *Proceedings of International Symposium on Computer Architecture*, pages 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. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, pages 123–132, March 2003.
- [29] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Reducing disk power consumption in servers with DRPM. *IEEE Computer*, pages 41–48, December 2003.

- [30] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Principles and Practice of Parallel Programming*, pages 186–195, Jun 2005.
- [31] Taliver Heath, Eduardo Pinheiro, Jerry Hom, Ulrich Kremer, and Ricardo Bianchini. Application transformations for energy and performance-aware device management. In *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques*, September 2002.
- [32] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking*, pages 130–142, 1996.
- [33] Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-power digital design. In *Symposium on Low Power Electronics*, pages 8–11, October 1994.
- [34] C-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *ACM SIGPLAN Conference on Programming Languages, Design, and Implementation*, June 2003.
- [35] Chung-Hsing Hsu and Wu-chun Feng. Effective dynamic-voltage scaling through CPU-boundedness detection. In *Fourth IEEE/ACM Workshop on Power-Aware Computing Systems*, December 2004.
- [36] Chung-Hsing Hsu and Wu-chun Feng. Towards efficient supercomputing: Choosing the right efficiency metric. In *First Workshop on High-Performance, Power-Aware Computing*, April 2005.
- [37] Chunling Hu, Daniel Jimenez, and Ulrich Kremer. Toward an evaluation infrastructure for power and energy optimizations. In *First Workshop on High-Performance, Power-Aware Computing*, April 2005.
- [38] M. Huang, J. Renau, and J. Torellas. Positional adaptation of processors: Application to energy reduction. In *International Symposium on Computer Architecture*, pages 157–168, June 2003.
- [39] M. Huang, J. Renau, and J. Torrellas. Profile-based energy reduction in high-performance processors. In *Proceedings of the 4th Workshop on Feedback-directed and Dynamic Optimizations*, December 2001.
- [40] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.
- [41] Nandani Kappiah, Vincent W. Freeh, and David K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In *Supercomputing*, November 2005.
- [42] Ken Kennedy and Ulrich Kremer. Automatic data layout for distributed-memory machines. *ACM Transactions on Programming Languages and Systems*, 20(4):869–916, 1998.
- [43] Ramakrishna Kotla, Soraya Ghiasi, Tom Keller, and Freeman Rawson. Scheduling processor voltage and frequency in server and cluster systems. In *Workshop on High-Performance, Power-Aware Computing (HPPAC)*, 2005.
- [44] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Mobicom 2002*, Atlanta, GA, September 2002.
- [45] R. Kravets, K. Schwan, and K. Calvert. Power-aware communication for mobile computers. In *Proc. 6th International Workshop on Mobile Multimedia Communications*, Nov 1999.

- [46] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Architectural Support for Programming Languages and Operating Systems*, pages 105–116, 2000.
- [47] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commercial servers. *IEEE Computer*, pages 39–48, December 2003.
- [48] Dong Li and Jun Wang. Eeraid: Energy efficient redundant and inexpensive disk array. In *European SIGOPS Workshop*, September 2004.
- [49] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *USENIX Winter*, pages 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 Institute of Technology, 2001.
- [51] Margaret Martonosi, David Brooks, and Pradip Bose. Modeling and analyzing CPU power and performance: Metrics, methods, and abstractions. In *SIGMETRICS*, 2001.
- [52] WattsUp Meters. <http://www.doubleed.com>.
- [53] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, 2002.
- [54] Orion Multisystems. <http://www.orionmulti.com/>.
- [55] Subbarao Palacharla, Norman P. Jouppi, and J.E. Smith. Complexity-effective superscalar processors. In *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, pages 206–218. ACM Press, 1997.
- [56] Feng Pan. Exploring the energy-time tradeoff in high performance computing. Master's thesis, North Carolina State University, Raleigh, NC 27695, May 2005.
- [57] Feng Pan, Vincent W. Freeh, and Daniel M. Smith. Exploring the energy-time tradeoff in high performance computing. In *High-Performance, Power-Aware Computing workshop*, in conjunction with IPDPS, Denver, CO, April 2005.
- [58] Athanasios E. Papathanasiou and Michael L. Scott. Energy efficiency through burstiness. In *WMCSA*, October 2003.
- [59] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '98*, pages 76–81, August 1998.
- [60] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*, September 2001.
- [61] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th Annual International Conference on Supercomputing*, pages 68–78, June 2004.
- [62] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power*, September 2001.

- [63] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Dynamic cluster reconfiguration for power and performance. pages 75–93, 2003.
- [64] J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISPLED '01*, August 2001.
- [65] M. Schulz, B. White, S. McKee, H. Lee, and J. Jeitner. Owl: Next-generation system monitoring. In *ACM Computing Frontiers*, pages 116–124, May 2005.
- [66] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, and Kevin Skadron. Power-aware QoS management in web servers. In *24th Annual IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.
- [67] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.
- [68] Robert C. Springer IV, David K. Lowenthal, Barry Rountree, and Vincent W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *Proceeding of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, March 2006.
- [69] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. *SIGOPS European Workshop*, 2000.
- [70] M. Valluri and L.K. John. Is compiling for performance = compiling for power? In *Proceedings of the 5th Annual Workshop on Interaction between Compilers and Computer Architectures*, 2001.
- [71] J. S. Vetter. Performance analysis of distributed applications using automatic classification of communication inefficiencies. In *International Conference on Supercomputing*, pages 245–254, May 2000.
- [72] M. Warren, E. Weigle, and W. Feng. High-density computing: A 240-node beowulf in one cubic meter. In *Supercomputing 2002*, November 2002.
- [73] Andreas Weissel, Bjorn Beutel, and Frank Bellosa. Cooperative IO - a novel IO semantics for energy-aware applications. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.
- [74] John Wilkes. Predictive power consumption. Technical Report HPL-CSP-92-5, Hewlett-Packard Labs, Feb 1992.
- [75] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler. Architectural requirements and scalability of the NAS parallel benchmarks. In *Proceedings of Supercomputing '99*, Portland, OR, November 1999.
- [76] Haijin Yan, Rupa Krishnan, Scott A. Watterson, David K. Lowenthal, Kang Li, and Larry L. Peterson. Client-centered energy and delay analysis for TCP downloads. In *14th IEEE International Workshop on Quality of Service*, June 2004.
- [77] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy: Unifying policies for resource management. In *USENIX 2003 Annual Technical Conference*, June 2003.

- [78] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: helping disk arrays sleep through the winter. In *Symposium on Operating Systems Principles*, pages 177–190, October 2005.
- [79] Qingbo Zhu, Francis M. David, Christo Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. Reducing energy consumption of disk storage using power-aware cache management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA-10)*, February 2004.