Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster[*]

Vincent W. Freeh
Feng Pan
Nandini Kappiah

Department of Computer Science
North Carolina State University
Raleigh, NC, USA

{vwfreeh,fpan2,nkappia}@ncsu.edu

David K. Lowenthal

Department of Computer Science
The University of Georgia
Athens, GA, USA

dkl@cs.uga.edu

ABSTRACT

Recently, system architects have built low-power, high-performance clusters, such as Green Destiny. The idea behind these clusters is to improve the energy efficiency of nodes. However, these clusters save power at the expense of performance. Our approach is instead to use high-performance cluster nodes that are frequency- and voltage-scalable; energy can than be saved by scaling down the CPU. Our prior work has examined the costs and benefits of executing an entire application at a single reduced frequency.

This paper presents a framework for executing a single application in several frequency-voltage settings. The basic idea is to first divide programs into *phases* and then execute a series of experiments, with each phase assigned a prescribed frequency. During each experiment, we measure energy consumption and time and then use a heuristic to choose the assignment of frequency to phase for the next experiment.

Our results show that significant energy can be saved without an undue performance penalty; particularly, our heuristic finds assignments of frequency to phase that is superior to any fixed-frequency solution. Specifically, this paper shows that more than half of the NAS benchmarks exhibit a better energy-time tradeoff using multiple gears than using a single gear. For example, IS using multiple gears uses 9% less energy and executes in 1% *less* time than the closest single-gear solution. Compared to no frequency scaling, multiple gear IS uses 16% less energy while executing only 1% longer.

Classification: D Software; D.4 Operating Systems; D.4.8 Performance.
General terms: Measurement, Performance.
Keywords: high-performance computing, power-aware computing.

1. INTRODUCTION

Recently, power-aware computing has gained traction in the high-performance computing (HPC) community. As a result, low-power, high-performance clusters, such as Green Destiny [64], have been developed to stem the ever-increasing demand for energy. Such systems improve the energy efficiency of nodes. Consider the case of Green Destiny—a cluster of Transmeta processors—which consumes less energy than a conventional supercomputer. In particular, Green Destiny 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) [64]. A reduction in performance by such a factor surely is unreasonable from the point of view of many users. If performance is the only goal, then one should continue on the current "performance-at-all-costs" path of HPC architectures. On the other hand, if power is paramount, then one should use a low-performance architecture that executes more instructions per unit energy.

This paper strikes a path between these two extremes: It uses a high-performance microprocessor that has frequency- and voltage-scaling. Each frequency provides an energy-performance point that we call a *gear*. It is conceptually possible to save energy without an increase in 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—usually waiting for memory, disk, or communication.

Consequently, there are opportunities where energy can be saved by reducing CPU frequency, without an undue performance penalty. On the other hand, during those parts of the program where the time penalty is large, the processors should be run at the fastest frequency.

Accordingly, this paper investigates energy savings that can be achieved, along with the corresponding execution time increase that results, from changing the microprocessor gear at different points in the program. As our experimental platform, we use a real *power-scalable cluster*, which in our case is a cluster composed of AMD Athlon-64 processors that are each frequency scalable—so their clock speed and hence power consumption can be changed dynamically.

We implement our scheme by first manually dividing programs into *phases* based on trace data collected during a profile of the application. If there are n phases in an application and g possible gears, there are g^n unique *solutions*, where a solution is an assignment of a gear to each phase. We present a novel heuristic that

searches the space of solutions. It executes a solution and evaluates the energy-time tradeoff based on a user-defined metric. Based on the evaluation, it selects the next solution to evaluate. The heuristic terminates when the results are satisfactory (again based on input from the user or cluster administrator). It completes in a linear number of steps based on the number of phases.

Performance results on the NAS suite show that our heuristic finds effective solutions. Specifically, we find several solutions that use different gears per phase that are superior to any solution that used a single gear for all phases. For example, for MG, one multiple-gear solution executes in the same time but consumes 3% less energy then a single-gear solution. This same multiple-gear solution executes 8% faster than another single gear while using the same amount of energy.

The **key contribution** of this paper is that it demonstrates that significant potential exists for energy savings in HPC applications without an undue increase in execution time. Therefore, it shows that changing gears during program execution is feasible and advantageous.

The rest of this paper is organized as follows. Section 2 describes related work. Next, Section 3 discusses our profile-directed technique, followed by the presentation of our measured results on our power-scalable cluster in Section 4. Finally, Section 5 summarizes and describes future work.

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 [9], 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 [54, 55]. Elnozahy *et al.* [18] investigated the policy in [54] as well as several others in a server farm. They found that when each node independently sets its voltage, the performance was almost as good as more complicated schemes that required coordination between server nodes. Such work shows that power and energy management are critical for commercial workloads, especially web servers [4, 41]. Additional approaches have been taken to include DVS [17, 61] and request batching [17]. The work in [61] 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 HPC effort that addresses the memory bottleneck is given in [32]; however, this is a purely static approach.

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 [28, 29]. 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 [73]. An alternative is to use an approach based on inspection of the program counter [23]; the basic idea is to infer the access pattern based on inspection of the program counter and shut down the disk accordingly. A final approach is to try to aggregate disk accesses in time; *i.e.*, if there are a total of N disk accesses separated by an average of T time units, the idea is to transform the program to have $M < N$ accesses separated by a time $T' > T$. This way the disk can be transitioned to a lower energy state for a longer period of time. A compiler/run-time approach using this was designed and implemented in [30], and a prefetching approach in [50]. Both were designed for mobile systems but can be directly applied to server/desktop systems.

The above transformations/optimizations are aimed at server farms. However, decreasing disk speed will decrease performance, so it cannot be applied arbitrarily. In addition, improved caching policies are unlikely to improve out-of-core HPC applications, because they tend to stream through the disk cache. Finally, increasing burstiness by aggregating disk requests in out-of-core applications is difficult due to limited memory.

There are also a few high-performance computing 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 [64], which uses low-power Transmeta nodes. A related approach is the Orion Multisystem machines [48], though these are targeted at desktop users. The latter two approaches sacrifice performance in order to save energy by using less powerful machines.

Finally, our prior work was an evaluation-based study that focused on exploring the energy/time tradeoff in the NAS suite [22]. Specifically, we found that using a *single* slower gear was in some cases able to save energy with little time delay. This work extends this idea by investigating the usefulness of varying the gear per phase, and it also adds an algorithm for choosing the assignment of gear to phase.

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.

At the system level, there is work in trying to make the OS energy-aware through making energy a first class resource [63, 16, 11]. Our approach differs in that we are concerned with saving energy in a *single* program, not a set of processes. One important avenue of application-level research on mobile devices focuses on collaboration with the OS [46, 65, 70, 71, 72, 21, 20, 2]. Such application-related approaches are complementary to our approach.

In terms of research on device-specific energy savings, there is work in the CPU via dynamic voltage scaling [19, 26, 52, 57, 25, 27, 34, 39, 44, 51, 53, 56, 58, 43, 49], the disk via spindown (*e.g.*, [31, 15, 66, 3, 42]), and on the memory or network [13, 40, 10, 36, 69, 68, 7, 8, 12, 37, 62, 38]. 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.

3. PROFILE-DIRECTED TECHNIQUE

This section describes our profiling technique for determining an effective gear for each phase. First, it describes detecting and prioritizing phases. Next, it discusses the mechanism that collects performance data and the energy-time tradeoff. Finally, it discusses our method for choosing an assignment of gear to each phase.

Phase Detection. This paper, uses a straightforward programming model, which primarily applies to iterative and predictable HPC applications. Specifically, it starts by obtaining a trace of the
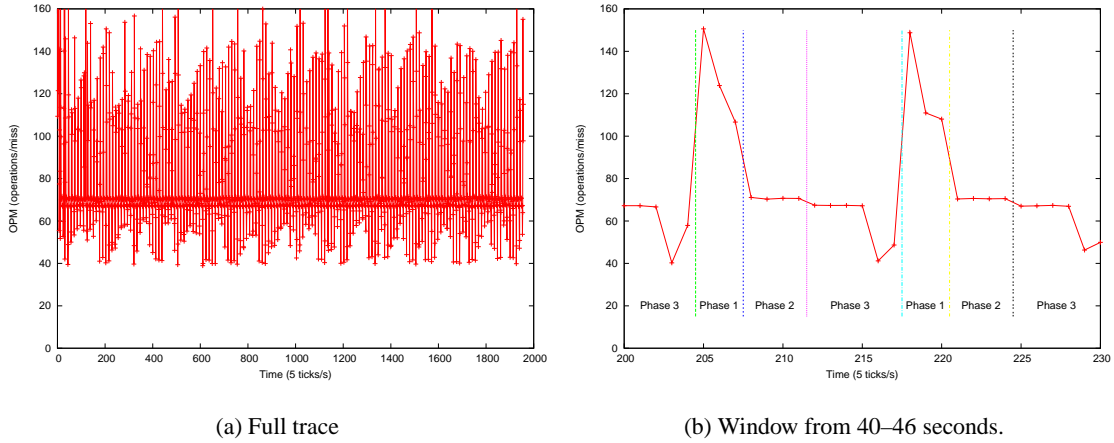
(a) Full trace

(b) Window from 40–46 seconds.

Figure 1: Trace of operations per miss for LU C.

(a) BT B
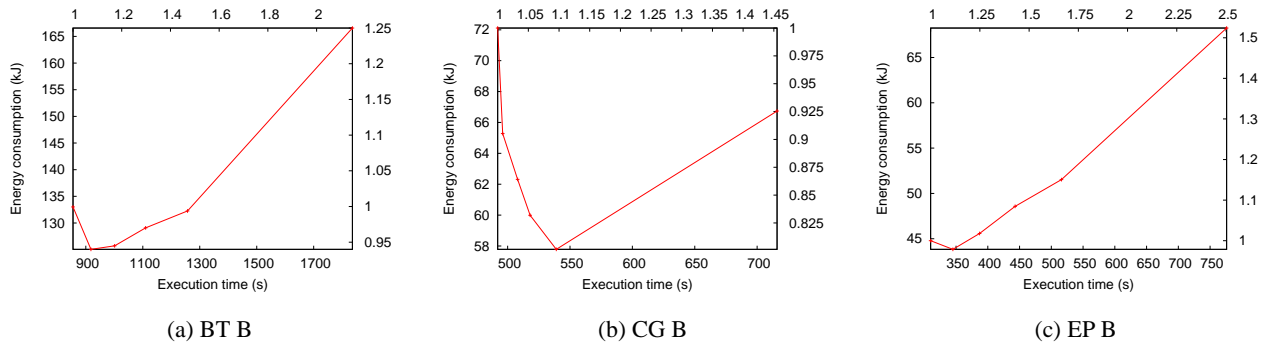

(b) CG B

(c) EP B

Figure 2: Energy consumption vs. execution time for three NAS benchmarks on a single AMD machine.

application in question (run at the fastest gear—highest frequency-voltage setting). From there, it divides a program into *blocks*. The term *block* is motivated by the common compiler term *basic block*. In a basic block, all statements must be executed. In a similar way, in a *block*, all statements execute at the same gear. (Beyond that, no similarity between these terms is implied.)

The actual division into blocks is done by examining the trace and using an ad hoc approach that conforms to the following principles. First, any MPI operation demarcates a block boundary. Second, if the *memory pressure* changes abruptly, a block boundary occurs at this change. (This is explained further below.) For the latter rule, the actual boundary is inserted at the end of the previous loop nest, following the notion that ends of loop nests are where program characteristics generally change [35]. We describe our approach assuming that there is an outermost loop that consists of a linear list of blocks (*i.e.*, no control flow). While either restriction can be relaxed, doing so unnecessarily complicates this discussion. Moreover, HPC codes often have one outermost ("timestep") loop, and MPI calls are generally unconditionally invoked.

From there, we merge blocks into *phases*. Two adjacent blocks are merged into a phase if their corresponding memory pressure is within the same threshold used in rule two above. We note that this is a simple algorithm and may not find all possible phases; our focus is not on developing new phase detection techniques. Our plan in the future is to use the significant work on phase detection that

has previously been done—both statically [35, 60, 47] and dynamically [33, 14].

We introduce the metric *operations per miss* (OPM) as a measure of the memory pressure of an application or phase. We have found OPM to be effective in determining phase boundaries. For example, Figure 1 gives an example of how OPM varies in the LU C benchmark on one of 8 nodes. The right-hand figure shows a window of 6 seconds. Here, OPM clearly partitions the code into three distinct phases. These phases were determined by hand, but it could be automated.

Phase Prioritization. Our method for assigning gears to phases requires ordering the phases such that the most likely phases to benefit from running at a lower gear are identified. This allows our algorithm to run in linear time, yet still find a desirable solution. Thus, our approach requires distinguishing a phase that has a good energy-time tradeoff from one that does not. That is, we must estimate the effect executing a phase in a slower gear will have on the energy consumption and execution time of the block. The key here again is OPM (introduced above), which estimates the phases that have a good energy-time tradeoff.

Figure 2 (previous page) shows the results of executing 3 NAS programs on a single Athlon-64 processor, and Table 1 shows the OPM for each of the programs. See [22] for a complete discus-

	EP	BT	LU	MG	SP	CG
OPM	844	79.6	73.5	70.6	49.5	8.60
Slope $0 \rightarrow 1$	-0.189	-0.811	-1.78	-1.11	-5.49	-11.7
Slope $1 \rightarrow 2$	0.288	0.0510	-0.355	-0.161	-1.52	-1.69

Table 1: Relating OPM to energy-time tradeoff.

sion. For each graph, the *total system energy* consumed at each gear (gear 0 is fastest, gear 5 is slowest) is plotted on the *y*-axis and the total execution time is plotted on the *x*-axis. The higher of 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. For readability, the origin of the graphs is not $(0, 0)$. Therefore, the alternate axes show the time and energy relative to the top gear (leftmost point).

EP is CPU bound and therefore has a large time penalty and little or no energy savings at reduced gears. On the other hand, CG is memory bound, so the time penalty is almost nothing. BT is a more typical application that sits between the extremes of EP and CG.

In Table 1, the benchmarks are sorted from highest to lowest OPM, ranging from a high of 844 for EP to a low of 8.60 for CG. The middle row shows the slope of the energy-time curve from top gear to gear 1, computed as $\frac{E_0 - E_1}{T_0 - T_1}$. A large (greater in magnitude) negative number indicates a near vertical slope and a significant energy savings relative to the time delay. On the other hand, a small number (positive or negative) indicates a near horizontal slope and little energy savings. Except for MG, the slopes in the table are sorted, in this case from greatest to least. Because a more negative slope indicates a better energy-time tradeoff, this table shows that memory pressure tends to predict the energy-time tradeoff.

Data Collection. The first step gathers profile data during an execution of the program. This implementation uses our *MPI-jack* tool, which is an interface that exploits PMPI [59], the profiling layer of MPI. *MPI-jack* enables a user transparently to intercept (hijack) any MPI call. A user can execute arbitrary code before and/or after an intercepted call. These are called *pre* and *post* hooks. In this work, we use *MPI-jack* to shift gears in post hooks.

The first step involves sampling an application, producing profile or trace data. An application is sampled at every phase boundary. This is done entirely through *MPI-jack*; in the case of MPI calls, it is trivial to add the sampling code. When it is necessary to insert a phase boundary at the end of a loop nest (*i.e.*, not at the end of an MPI call), we simply insert a *pseudo* MPI call; this way, we handle all phases in a uniform way.

The information we collect includes the type of call and location (program counter). It shows status (gear, time, *etc.*) and metrics (μops and L2 cache misses, so OPM can be computed). Note that our inclusion of the program counter bears some resemblance to existing program counter based techniques [23], though that work is aimed solely at saving energy in the disk.

Energy-Time Tradeoff. Our analysis (see below) requires energy consumption data; however, we have found that energy cannot be measured accurately if the period of measurement is too fine grain.[1] For this paper, as the benchmark programs complete rela-

tively quickly, we measure energy at the coarsest grain possible—the complete run of a program. (For longer running programs we can terminate a test run after a fixed number of iterations or phases.)

Determining which of two solutions is "better" depends on how a user wants to trade off energy savings and time delay. This work does not impose an evaluation. Rather, it leaves it to the user or cluster administrator to select the metric. It could be energy-delay or energy-delay squared, the latter proposed in [45, 24], which has been adopted by Cameron *et al.* [5] for use in power-aware, high-performance computing.

Methodology. Given a program partitioned into phases, we proceed to our method for determining an effective assignment of gears to phases. If there are n phases and g gears, than the number of possible solutions (phase-gear assignments) for the program is g^n. In general, this is too large to explore by brute force. Therefore, the second part of our method is a heuristic (described in Figure 3) that we use to find the "best" solution. The heuristic finds the "best" gear in a phase, then moves on to the next phase. Once it moves on to another phase, the gear for the preceding phase has been determined. Therefore, it is important that phases are sorted.

Initially, the solution G (a vector of gears) is set to the baseline value—all zeroes. The recursive function is invoked on the 0th phase. It executes the program using the next slower gear in this phase (all other phases are as before). If the energy-time tradeoff (defined by a user relation, see below) of this new solution is better than the current solution, it is accepted. The algorithm recursively tries the next lower gear on this phase. The gear is determined when the new tradeoff is worse than the current or when there are no slower gears. After fixing the gear, it moves on to the next phase. This heuristic has running time at most $n \times g$.

After each program execution, the energy and time are measured and compared via the user-defined relationship. For our tests, we use a simple and intuitive evaluation of the tradeoff based on the *slope* of the line between two solutions. The slope is defined as the ratio of energy savings to time delay: $\frac{E_i - E_j}{T_i - T_j}$, for two solutions i and j. Thus a slope of -1 (*i.e.*, $45°$ below the horizon) means savings and delay are equally weighted. A user-defined limit of 0 means minimize energy, and $-\infty$ means minimize time. We consider a new solution with a larger slope (in magnitude) than the user-defined limit to be better. We do not advocate this metric instead of the others. We use it because it is reasonable and it is easy to visualize.

4. MEASURED RESULTS

This section describes the results of our experiments using our power-aware cluster. First, we give our experimental methodology. Second, using BT, we present an example of our methodology along with results. Third, we present overall results from all the NAS applications. Finally, we analyze the results.

4.1 Experimental Methodology

We studied the programs in the NAS parallel benchmark suite using either 8 or 9 nodes, depending on the benchmark. Presumably, such mature benchmarks have been thoroughly analyzed and

---

[1]This is in part a function of our measuring equipment. However, fine-grain measurement of system power is problematic because unpredictable high-power events, such as disk spin-up, skew such results. Larger time intervals tend to mask the effect of such events.

```
/* G is an n-dimensional vector of gear selections, one for each phase. */
set $G_k = 0, \forall k | 0 \leq k < n$    /* The top gear is 0; slowest gear is $n-1$ */
given $\mathcal{T}$    /* Tuple (energy,time) for current $G$ (initially the baseline) */
given $\prec$    /* A user-defined relationship that defines total order of $\mathcal{T}$ */
    /* Invoke the function to start method that produces final solution, $G_f$ */

$G_f \leftarrow$ evaluate(program, $G, 0, n, \mathcal{T}$)

define evaluate(program, $G, i, n, \mathcal{T}$)
    if $i \geq n$ or $G_i \geq g_{slowest}$ then return $G$ fi
    $G_i \leftarrow G_i + 1$
    execute program using solution $G$
    measure energy and time for $G$, store tuple in $\mathcal{T}'$
    if $\mathcal{T}' \prec \mathcal{T}$ then    /* $\mathcal{T}'$ is not better than $\mathcal{T}$ */
        $G_i \leftarrow G_i - 1$
        $G =$ evaluate(program, $G, i + 1, n, \mathcal{T}$)
    else                    /* $\mathcal{T}'$ is better than $\mathcal{T}$ */
        $G =$ evaluate(program, $G, i, n, \mathcal{T}'$)
    fi
    return $G$
end
```

Figure 3: Heuristic for searching solution space.

are well-written (*e.g.*, see [67])—so that they are not unrealistically communication bound. Our approach capitalizes partly on blocking receive time waiting for data. Therefore, well-tuned programs like the NAS programs should result in an approximate lower bound in terms of saving energy during idle time resulting from communication.

Our experimental platform is a cluster of ten frequency- and voltage-scalable AMD Athlon-64s. Its available operating points are in the range of 800–2000MHz and 0.9–1.5V. Each node has 1GB main memory, a 128KB L1 cache (split), and a 512KB L2 cache, and the nodes are connected by 100Mb/s network. In this paper, we vary the CPU power and measure overall system energy. Although there are other components, throttling the CPU is effective in saving energy because the CPU is a major power consumer. In particular, the Athlon-64 CPU used in this study consumes approximately 45–55% of overall system energy.[2]

For each program we measure execution time and energy consumed. Execution time is elapsed wall clock time. The voltage and current consumed by the entire system is measured by precision multimeters at the wall outlet to determine the instantaneous power (in Watts). This value is integrated over time to determine the energy used. Integration is performed by a separate computer that samples the multimeters hundreds of times a second.

4.2   In-Depth Example of Methodology

This section gives an example of how our methodology works, along with results. For this purpose, we study BT class C in depth. In the graph shown in Figure 4, the *baseline* is the leftmost (fastest) point, where the top gear, 2000MHz, is used for the entire program. (As before, the higher of two points uses more energy, and the further right of two points takes more time.) For all other points, at least one phase is executed in a lower gear. Each point is labeled as a tuple, where the ith entry represents the gear used during the ith phase. (In general, applications may have different size tuples,

---

[2]CPU power is not measured directly. However, the system power at the fastest energy gear is 145–160 W. The AMD datasheet states that the absolute maximum CPU power dissipation is 89 W. We estimate the peak power of the CPU for our application is in the range of 70–80 W, which is roughly 45–55% of system power.

because they have a different number of phases.) For convenience, we refer to a tuple of phase-gear assignments as a *solution*.

Our analysis of the operations per miss (OPM) identified two phases in BT. The baseline solution is labeled "00", meaning both phases are run in gear 0. In the figure we have drawn a line connecting 5 points that are "good" choices under a simple slope-based energy-time metric. The line forms a convex hull that is left of and below all other solutions. Any solution "inside" the hull is not a "good" choice according to this metric, although there may exist solutions inside the hull that are good under a different metric.

Recall that the goal of our profiling algorithm is to do "better" than the baseline. As our chosen metric is the slope between two solutions, $\mathcal{L}$, our algorithm will select a unique point on the hull. This is illustrated in Table 2, which shows five different cases regarding BT.
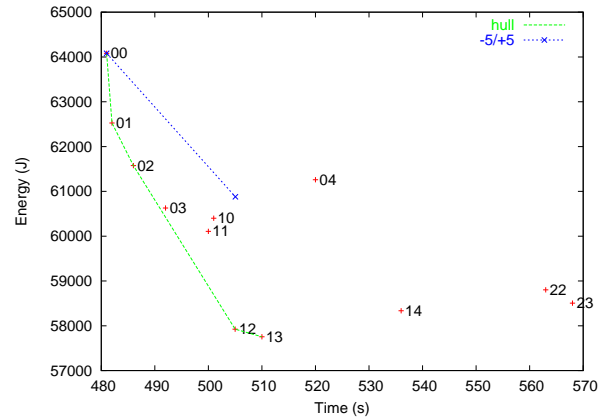

Figure 4: Energy-time plot of several BT runs.

In the first case (Table 2(a)), $\mathcal{L}$ is steep (large negative number). Such a value favors time delay over energy savings. In this case and all others, the algorithm adjusts gears from right to left across solutions or tuples. Thus, the first test is solution 01. The steps of the algorithm are shown in the table. The second column shows the

	Solutions	Slope	< $\mathcal{L}$?
1	00 → 01	-11.7	false
2	00 → 10	-1.39	false
	00 is best		

(a) Case 1: $-12 > \mathcal{L}$.

	Solutions	Slope	< $\mathcal{L}$?
1	00 → 01	-11.7	true
2	01 → 02	-1.78	false
3	01 → 11	-1.01	false
	01 is best		

(b) Case 2: $-1.78 > \mathcal{L} > -12$.

	Solutions	Slope	< $\mathcal{L}$?
1	00 → 01	-11.7	true
2	01 → 02	-1.78	true
3	02 → 03	-1.19	false
4	02 → 12	-1.44	false
	02 is best		

(c) Case 3: $-1.44 > \mathcal{L} > -1.78$.

	Solutions	Slope	< $\mathcal{L}$?
1	00 → 01	-11.7	true
2	01 → 02	-1.78	true
3	02 → 03	-1.19	false
4	02 → 12	-1.44	true
5	12 → 22	-0.25	false
	12 is best		

(d) Case 4: $0 > \mathcal{L} > -1.44$.

	Solutions	Slope	< $\mathcal{L}$?
1	00 → 01	-11.7	true
2	01 → 02	-1.78	true
3	02 → 03	-1.19	true
4	03 → 04	0.17	false
5	03 → 13	-1.20	true
6	13 → 23	0.97	false
	13 is best		

(e) Case 5: $\mathcal{L} = 0$.

Table 2: Five cases for BT.

two solutions being compared, and the arrow indicates the direction of the slope. Of these two solutions, the energy and time of the one on the left is known, so only one run of the program is necessary to complete this step. The third column shows the slope, and the fourth column indicates whether is it less than the user-defined limit. In the first step, the slope from 00 to 01 is greater than $\mathcal{L}$ (not as steep); therefore, it is rejected. We back up to the previous solution (00) and try the next phase to the left (which in this case happens to be the leftmost phase). In step 2, solution 10 is rejected. Thus, the algorithm selects 00, the baseline, when $\mathcal{L}$ is sufficiently steep.

Table 2(b) shows the steps in the second case. In this case, 01 is accepted but 02 is rejected. The algorithm fixes the second phase at gear 1 and then determines the gear for the first phase, ultimately selecting 01. The next case (Table 2(c)) is similar; the slope limit is slightly larger (less steep), which results in acceptance of 02, followed by the rejection of 03 and 12.

Table 2(d) shows the steps in case 4. In this case, the second phase is fixed at gear 2. The solution for 12 is considered better than 02, so the algorithm accepts it and tries 22, which is rejected.

The last case is shown in Table 2(e). This case temporarily selects solution 03, which is not on the convex hull, before eventually selecting 13. The solution 03 is interesting, because the algorithm and evaluation metric we describe here will never select it. This is because our slope metric only picks solutions that are on the convex hull. However, 03 is a reasonable solution in that there does not exist a solution that is better than it in both time and energy (*i.e.*, "dominates" it). So, it easily can be argued that 03 is better than 02. For BT, solutions 03 and 11 fall into this category and therefore may be selected by a *different* evaluation metric. However, the other solutions (04, 10, 14, 22, and 23) will never be selected by any metric, because each is dominated in both energy and time by at least one other solution.

4.3 Analysis of Results

Figure 5 shows results from the other six NAS programs that we executed. Because each graph has different *x* and *y* ranges, we plot

a normalizing line that shows a 5% decrease in energy and a 5% increase in time. It is important to consider both the direction and magnitude of this line. Overall, the behavior of the applications varies widely. Each program falls into one three groups based on its benefit from using a reduced gear. In each figure, we show the convex hull along with several points that were tested while executing the algorithm.

Multiple Gear Benefit. Applications in this group show significant benefit from a multiple-gear solution. This is the case whenever several ij solutions, where $i \neq j$, fall on the convex hull. Four of the seven NAS programs have this characteristic: MG, BT, LU, and IS. For example, in multiple-gear solution 32, MG saves 11% energy with a 4% time penalty over the baseline. On the other hand, the single-gear solution 33 saves 10% energy with a 7% time delay. BT using solution 12 saves 10% energy with a 5% time penalty over the baseline. This compares favorably to single gear solutions 11 and 22, which yield energy-time tradeoffs of -6%/4% and -8%/17%, respectively.

In the case of LU, there are three phases, as shown above in Figure 1. This program is the only one for which we identified more than two phases. This plot has more points than any other plot, yet 70 more points are needed to exhaustively search just the top three gears (0 thru 2).

IS is an extreme case, where the first phase is CPU bound and the second phase is both memory *and* disk bound. Therefore, a single-gear solution is bound to be poor, as it is necessarily a compromise solution. Regardless of the desire of the user, single-gear solutions (other than the baseline) are dominated by points on the hull. The 05 solution in IS saves 16% energy over the baseline (00) at a cost of a 1% time increase. Compared instead to 22, this solution saves 9% energy and executes 1% faster.

Single Gear Benefit. Applications in this group show significant benefit from using a single lower gear, but no significant benefit from multiple-gear solutions. CG and SP fall into this category; however, for different reasons. In CG, only one phase was detected.
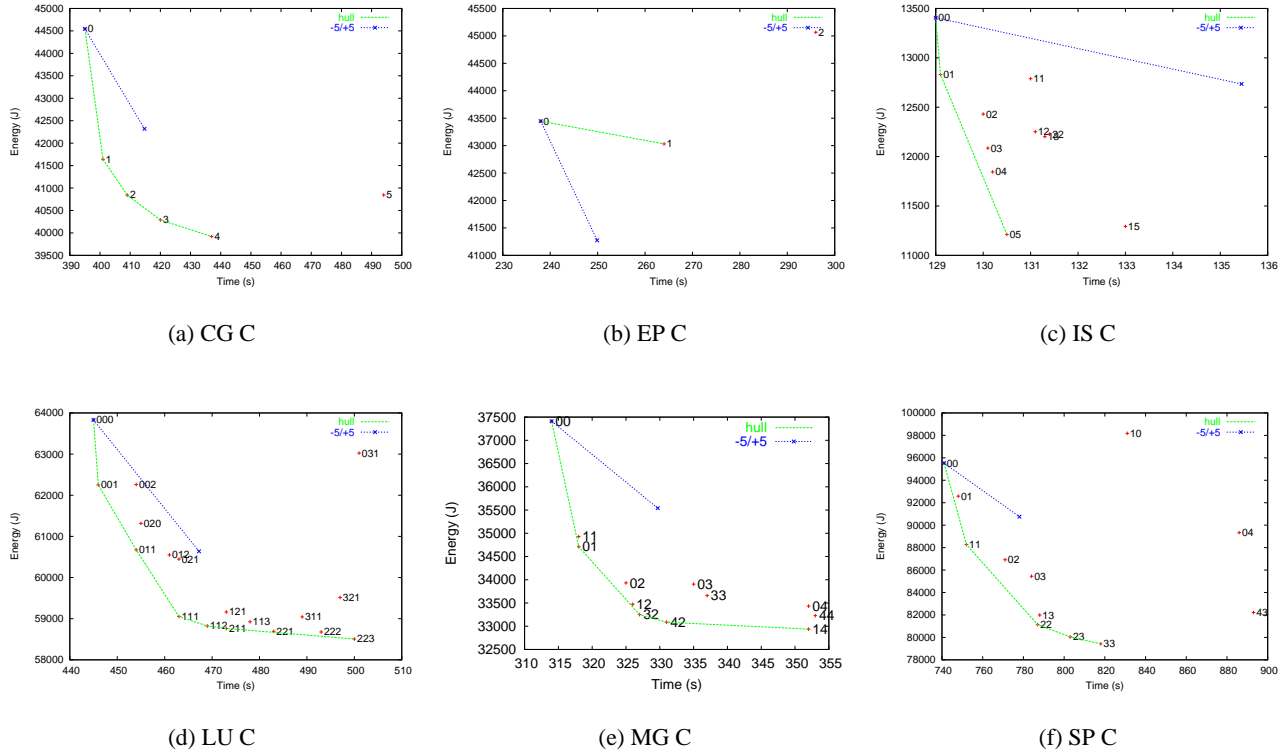
Figure 5: Energy consumption vs. execution time for NAS class C benchmarks 8 or 9 nodes.

The application is highly memory bound: its is OPM an order of magnitude smaller than any other. Moreover, all blocks in the loop have comparable OPM.

In SP, we while we found two phases, there is almost no benefit to running phases in different gear. Although 23 is on the convex hull, it barely makes it. SP illustrates that it is important to sort the phases. Searching from the first phase, our method tries 10 which is rejected, fixing the first phase in gear 0. Thus, the method would select one of 00, 01, 02, or 03. All but the baseline are not on the convex hull. In particular, suppose the user set $\mathcal{L}$ to -3. Searching in the wrong order yields solution 01. However, searching from the second phase, this method tries 01, which passes, then 02, which fails. Then it moves on to the first, tries 11 then 21, which it rejects. Thus it selects 11.

Besides the two general reasons given above, there is a third reason that a single gear benefit is all that can be achieved. Conceptually, an application can fall into the single gear category because the phases are too fine-grain (possibly because there are too many) and the cost of switching gears outweighs the benefit. However, this was not the case in any of the NAS programs.

No Benefit. Only EP falls into this category. Simply put, EP has only one phase, no communication (except once at the end of the program), and is CPU bound. Therefore, a reduction in the gear results in the program taking longer to complete and has little or no energy savings.

5. CONCLUSIONS AND FUTURE WORK

This paper investigates energy savings in HPC applications, along with the corresponding time delay, that results from varying the processor gear per program phase. Given an application, a program is divided into *blocks* which are merged into *phases* using program traces. A novel heuristic successively evaluates *solutions* (phase-gear assignments). The program is executed, and energy and time are measured. Then, the heuristic evaluates the solution based on a user-defined metric and selects the subsequent solution to evaluate. Performance results were obtained on the NAS suite on a real *power-scalable cluster* composed of AMD Athlon-64 frequency and voltage scalable processors. We found that in most NAS benchmarks, using multiple gears in a single application can provide a better energy-time tradeoff over any single-gear solution.

Future Work. There are many avenues we are investigating that relate to this work. First, we will enhance our profile-directed technique. With a small number of phases, our current method suffices; however, for programs with large numbers of phases, even the linear-time algorithm discussed in this paper may be too costly. Second, we will consider *inter-node bottlenecks*, where a subset of the nodes reach a synchronization point later than the rest of the nodes. A node bottleneck can occur for a variety of reasons, but the end result is that early-arriving nodes can be scaled down with little or no performance degradation. This necessarily means that we will have to allow different gears on different nodes, in the same phase. Third, we intend to automate our entire method, including phase detection, as much as possible. Fourth, we will incorporate I/O into our intra-node bottleneck evaluation. Finally, we are starting to experiment with large-scale programs; while we believe the NAS programs are representative, they are not industrial-strength codes and hence have only a few phases.

6. REFERENCES

[1] N.D. Adiga et al. An overview of the BlueGene/L supercomputer. In *Supercomputing 2002*, November 2002.

[2] Manish Anand, Edmund Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Mobicom*, September 2003.

[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] 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.

[5] Kirk W. Cameron, Rong Ge, Xizhou Feng, Drew Varner, and Chris Jones. High-performance, power-aware distributed computing framework (poster). In *Supercomputing 2004*, November 2004.

[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] S. Chandra and A. Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *USENIX Annual Technical Conference*, 2002.

[8] Surendar Chandra. Wireless network interface energy consumption implications of popular streaming formats. In *Multimedia Computing and Networking (MMCN '02)*, Jan 2002.

[9] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centres. In *Symposium on Operating Systems Principles*, pages 103–116, 2001.

[10] IEEE Computer Society LAN/MAN Standards Committee. IEEE Std 802.11: Wireless LAN medium access control and physical layer specification. Technical report, August 1999.

[11] Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface specification, revision 2.0. July 2000.

[12] A. Datta, A. Celik, J. G. Kim, D. E. VanderMeer, and V. Kumar. Adaptive broadcast protocols to support power conservant retrieval by mobile users. In *ICDE*, pages 124–133, 1997.

[13] 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.

[14] A. Dhodapkar and J. Smith. Comparing phase detection techniques. In *International Symposium on Microarchitecture*, pages 217–227, December 2003.

[15] F. Douglis, 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.

[16] C.S. Ellis. The case for higher-level power management. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*, March 1999.

[17] Elmootazbellah Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *USITS '03*, 2003.

[18] E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Workshop on Mobile Computing Systems and Applications*, Feb 2002.

[19] 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.

[20] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, 1999.

[21] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.

[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] 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.

[24] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. In *IEEE International Symposium on Low Power Electronics*, October 1995.

[25] K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Mobile Computing and Networking*, pages 13–25, 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] D. Grunwald, P. Levis, K. Farkas, C. Morrey, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of 4th Symposium on Operating System Design and Implementation*, October 2000.

[28] 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.

[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, 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.

[31] 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.

[32] 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.

[33] 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.

[34] C. Im, H. Kim, and S. Ha. Dynamic voltage scheduling technique for low-power multimedia applications using buffers. In *Proceedings of the International Symposium on*

Low-Power Electronics and Design ISPLED '01, August 2001.

[35] Ken Kennedy and Ulrich Kremer. Automatic data layout for distributed-memory machines. *ACM Transactions on Programming Languages and Systems*, 20(4):869–916, 1998.

[36] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Mobicom 2002*, Atlanta, GA, September 2002.

[37] R. Kravets and P. Krishnan. Power management techniques for mobile communication. In *Proceedings of the 4th International Conference on Multimedia Computing and Networking*, pages 157–168, Oct 1998.

[38] R. Kravets, K. Schwan, and K. Calvert. Power-aware communication for mobile computers. In *Proc. 6th International Workshop on Mobile Multimedia Communications*, Nov 1999.

[39] C. M. Krishna and Y. H. Lee. Voltage-clock-scaling techniques for low power in hard real-time systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 156–165, May 2000.

[40] 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.

[41] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commerical servers. *IEEE Computer*, pages 39–48, December 2003.

[42] 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.

[43] J. Lorch and A. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications Magazine*, 5(3):60–73, June 1998.

[44] J. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. In *Proceedings of the ACM SIGMETRICS 2001 Conference*, pages 50–61, June 2001.

[45] Margaret Martonosi, David Brooks, and Pradip Bose. Modeling and analyzing CPU power and performance: Metrics, methods, and abstractions. In *SIGMETRICS*, 2001.

[46] Robert J. Minerick, Vincent W. Freeh, and Peter M. Kogge. Dynamic power management using feedback. In *Workshop on Compilers and Operating Systems for Low Power*, pages 6–1–6–10, Charlottesville, Va, September 2002.

[47] Donald G. Morris and David K. Lowenthal. Accurate data redistribution cost estimation in software distributed shared memory systems. In *Principles and Practice of Parallel Programming*, pages 62–71, June 2001.

[48] Orion Multisystems. http://www.orionmulti.com/.

[49] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems and Principles*, pages 276–287, October 1997.

[50] Athanasios E. Papathanasiou and Michael L. Scott. Energy efficiency through burstiness. In *WMCSA*, October 2003.

[51] T. Pering and R. Brodersen. Energy efficient voltage scheduling for real-time operating systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS '98*, June 1998.

[52] 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.

[53] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[54] 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.

[55] 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.

[56] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th Conference on Mobile Computing and Networking MOBICOM '01*, July 2001.

[57] 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.

[58] Gang Quan and Xiaobo Hu. Energy efficient fixed priority scheduling for real-time systems on variable voltage processors. In *ACM/IEEE Design Automation Conference*, pages 828–833, June 2001.

[59] Rolf Rabenseifner. Automatic profiling of MPI applications with hardware performance counters. In *PVM/MPI*, pages 35–42, 1999.

[60] Umit Rencuzogullari and Sandhya Dwarkadas. Dynamic adaptation to available resources for parallel computing in an autonomous network of workstations. In *Eighth Conference on Principles and Practice of Parallel Programming*, pages 72–81, June 2001.

[61] 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.

[62] M. Stemm, P. Gauthier, D. Harada, and R. H. Katz. Reducing power consumption of network interfaces in hand-held devices. In *Proc. 3rd Intl. Workshop on Mobile Multimedia Comm.*, September 1996.

[63] 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.

[64] M. Warren, E. Weigle, and W. Feng. High-density computing: A 240-node beowulf in one cubic meter. In *Supercomputing 2002*, November 2002.

[65] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Operating Systems Design and Implementation (OSDI '94)*, pages 13–23, 1994.

[66] John Wilkes. Predictive power consumption. Technical Report HPL-CSP-92-5, Hewlett-Packard Labs, Feb 1992.

[67] 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.

[68] Haijin Yan, Rupa Krishnan, Scott A. Watterson, and David K. Lowenthal. Client-centered energy savings for concurrent HTTP connections. In *Proceedings of the 14th ACM Workshop on Networks and Operating System Support for Digital Audio and Video*, June 2004.

[69] 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 *Proceedings of the 14th IEEE International Workshop on Quality of Service*, June 2004.

[70] Giovanni De Micheli Yung-Hsiang Lu, Luca Benini. Operating-system directed power reduction. In *International Symposium on Low Power Electronics and Design*, pages 37–42. Stanford University, July 2000.

[71] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS X*, October 2002.

[72] 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.

[73] 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.