Power-Aware Speedup

Rong Ge and Kirk Cameron

Scalable Performance Laboratory
Department of Computer Science and Application
Virginia Tech, Blacksburg, VA 24061
(ge, cameron)@cs.vt.edu

ABSTRACT

Power-aware processors operate in various power modes to reduce energy consumption with a corresponding decrease in peak processor throughput. Recent work has shown power-aware clusters can conserve significant energy (>30%) with minimal performance loss (<1%) running parallel scientific workloads. Nonetheless, such savings are typically achieved using a priori knowledge of application performance. Accurate prediction of parallel power consumption and performance is an open problem. However, such techniques would improve our understanding of power-aware cluster tradeoffs and enable identification of "sweet spots" in system configurations optimized for performance and power. Speedup models are powerful analytical tools for evaluating and predicting the performance of parallel applications. Unfortunately, existing speedup models do not quantify parallel overhead for simplicity leaving them incapable of accurately accounting for performance and power. We propose *power-aware speedup* to model and predict the scaled execution time of power-aware clusters. The new model accounts for parallel overhead and predicts (within 7%) the power-aware performance and energy-delay products for various system configurations (i.e. processor counts and frequencies) on NAS Parallel benchmark codes.

1. INTRODUCTION

The electricity bills for multi-megawatt high-end systems, such as NASA Columbia or Earth Simulator, are in the hundreds of thousands of dollars annually. In just two hours, Earth Simulator could produce enough BTU's to heat an average 2000 square foot home in the U.S. Midwest all winter long. As newer, larger machines are designed, such costs and the increased failure rates due to elevated machine room temperatures cannot be allowed to continue unabated.

Power-aware computing has recently gained traction in the high-performance community [6, 7, 11-14, 16, 22-24]. Results from several groups of researchers have shown energy savings are possible using a priori performance profiling to identify communication-bound phases in parallel codes and reduce power to the processors by applying DVFS (dynamic voltage and frequency scaling) to these phases.

Accurate prediction of parallel power consumption and performance would improve our understanding of power-aware cluster tradeoffs and enable identification of "sweet spots" in system configurations optimized for performance and power. Nonetheless, such models for power-aware parallel systems do not exist presently.

Any power-aware model of parallel performance must accurately quantify the amount of execution time affected by power modes. For example, parallel overhead influences the percentage of total execution time affected by parallelism. Similarly, parallel overhead affects the percentage of total execution time affected by processor frequency. Furthermore, the percentage of total execution time due to parallel overhead changes with application and number of nodes. Thus the execution time effects of frequency and parallelism are interdependent. As we

will see in the next section, this complicates modeling the power-performance of power-aware clusters.

In this paper, *we address the problem of identifying a metric that models the performance of power-aware clusters.* We develop a new speedup model that quantifies the effects of power modes on workload and accurately identifies the affected workload portions. Currently, we focus on isolating the performance effects of changing processor frequencies (power) and the number of nodes (parallelism).

We identify several **key contributions of this work**:

- We introduce the power-aware speedup model to analytically capture the interacting effects of parallelism and processor frequency.
- We analyze the power-aware speedup for various scientific applications included in the NAS Parallel Benchmark suite running on a power-aware cluster.
- We show how to derive parameters for our model and use them to predict performance and the power-aware speedup for scientific applications.

The structure of the rest of the paper is as follows. In Section 2, we show a motivating example for this work. We present our power-aware speedup model in Section 3 and use it to analyze the speedups for NPB benchmarks on a power-aware parallel system in Section 4. In Section 5 we show how to predict application performance and speedup using simplified parameterization method and fine-grain parameterization method. Related work and conclusions are discussed in Sections 6 and 7 respectively.

2. MOTIVATING EXAMPLE

Amdahl's Law, or the law of diminishing returns, is a parallel speedup model commonly used by the research community. The basic idea is that any system enhancement is only applicable to a certain portion of a workload. For parallel computing, the increased number of nodes is considered the enhancement and the speedup is often defined as the ratio of sequential to parallel execution time:

$$S_N(w) = \frac{T_1(w)}{T_N(w)} \quad (1), \text{ where}$$

w: the *workload* or total amount of work (in instructions or computations),

$T_1(w)$: the *sequential execution time* or the amount of time to complete workload w on 1 processor, and

$T_N(w)$: the *parallel execution time* or the amount of time to complete workload w on N processors.

If the fraction of enhanced workload (FE) is the portion of the total workload that is parallelizable, and the enhancement will reduce execution time of the parallelizable workload portion by a speedup factor (SE), the parallel speedup for the entire workload can be expressed [1, 27] as:

$$S_N(w) = \frac{T_1(w)}{T_N(w)} = \left[(1 - FE) + \frac{FE}{SE} \right]^{-1} \quad (2).$$

For e enhancements where $e>=1$, we can generalize Equation 2 [27] as:

$$S_N(w) = \prod_e \left[(1 - FE_e) + \frac{FE_e}{SE_e} \right]^{-1} \quad (3).$$

Equation 3 states that the speedup for a workload using *e* simultaneous enhancements is the product of the individual speedups for each enhancement. This generalization of Amdahl's Law is the only available speedup model that considers multiple enhancements simultaneously. Thus, we investigate its suitability for power-aware parallel systems.

Table 1. To determine the best system configuration for FT for all combinations of frequency and processor count, we need pairwise speedup comparisons to the slowest frequency (600 MHz) and the smallest number of nodes (N=1) as the base sequential execution time. To predict speedup, we use Equation 3 for e=2 variables. Each table entry is the relative error. 600 MHz is used as the basis for comparison, so its column shows no error since it effectively varies only with number of nodes, exemplifying traditional speedup. Errors occur when trying to compare the results for two enhancements simultaneously since their effects are interdependent and not modeled by Equation 3.

N	Frequency (MHz)				
	600	800	1000	1200	1400
2	0%	30%	31%	49%	66%
4	0%	18%	36%	42%	58%
8	0%	30%	41%	59%	78%
16	0%	26%	40%	54%	72%

In power-aware clusters, a typical goal is to maximize performance while minimizing power consumption. Speedup models can be used to predict performance, and thus identify "sweet spot" system configurations of processor count and frequency that meet these constraints. If the performance or speedup prediction is accurate, we can either select the best speedup across all the data, or use the execution time predictions in an energy-delay metric [3] to determine the tradeoffs between performance and energy.

We use the speedup model in Equation 3 to predict the simultaneous effects of processor count and frequency on speedup relative to the lowest processor frequency (600 MHz) and smallest number of processors (*N=1*). Table 1 shows the speedup prediction errors of a parallel Fourier transform (FT) application. Due to the large errors between predicted and measured speedup, identifying "sweet spot" system configurations using Equation 3 for multiple enhancements is problematic.

Equation 3 over predicts speedup on power-aware clusters since it assumes the effects of multiple enhancements are independent. Power-aware clusters and applications violate this assumption since parallel overhead depends on processor counts and influences the effects of frequency scaling. Use of Equation 3 to model FT on a 16-node power-aware cluster gives errors as large as 78%, 45% on average.

In our power-aware cluster work, we combine the effects of processor count and frequency into a metric that captures and explains their simultaneous effects on execution time. Ultimately, we would like to predict these effects for unmeasured processor counts and frequencies. To this end, we propose power-aware speedup and denote it as:

$$S_N(w,f) = \frac{T_1(w,f)}{T_N(w,f)} \quad (4), \text{ where}$$

w: the *workload* or total amount of work (in instructions or computations),

f: the clock *frequency* in clock cycles per second,

$T_1(w,f)$: the *sequential execution time* or the amount of time to complete workload *w* on *1* processor for frequency *f*, and

$T_N(w,f)$: the *parallel execution time* or the amount of time to complete workload *w* on *N* processors for frequency *f*.

Power-aware speedup is the ratio of sequential execution time for a workload (*w*) and frequency (*f*) on *1* processor to the parallel execution time for a workload and frequency on *N* processors. In the next section we detail the additional equations necessary to quantify the execution times of Equation 4. In succeeding sections, we show how our model improves the

error rates shown in Table 1 and we identify the key differences between power-aware speedup and Equations 1-3 (Amdahl's Law).

3. POWER-AWARE SPEEDUP

In this section, our goal is to describe power-aware speedup in the simplest terms possible. We will use the terms defined by Equation 4 and introduce definitions as needed to understand each derivation step and then use the defined terms to express equations that build on one another. By the end, the equations will be quite large, but our hope is the fundamental concepts remain straightforward.

Sequential execution time for a single workload ($T_1(w,f)$)

CPI: the average number of clock *cycles per workload*.

Using this definition and others from Equation 4, we define sequential execution time as

$$T_1(w, f) = w\frac{CPI}{f} \quad (5).$$

This is a variant of the CPU performance equation [27]. The time to execute a program on *1* processor is the product of the workload (w) and the rate at which workloads execute (CPI/f or seconds per workload). For now, we assume f is a fixed value, noting that $T_1(w,f)$ depends on the processor frequency.

Sequential execution time for an ON-chip/OFF-chip workload ($T_1(w^{ON}, f^{ON})$, $T_1(w^{OFF}, f^{OFF})$)

w^{ON}: ON-chip workload, or all workloads that do not require data residing OFF-chip at the time of execution.

w^{OFF}: OFF-chip workload, or all workloads that require OFF-chip data accesses at the time of execution.

f^{ON} : ON-chip clock *frequency* in clock cycles per second. Affected by processor DVFS.

f^{OFF} : OFF-chip clock *frequency* in clock cycles per second. Not affected by processor DVFS.

CPI^{ON} , CPI^{OFF} : the average number of clock cycles per ON-chip (CPI^{ON}) or OFF-chip (CPI^{OFF}) workload.

Others have shown [8, 33] that a given workload (w) can be divided into ON-chip (w^{ON}) workload and OFF-chip (w^{OFF}) workload. Under these constraints, the total amount of work (in instructions or computations) is given as $w=w^{ON}+w^{OFF}$. We can modify our simple representation of sequential execution time[1] as:

$$T_1(w, f) = T_1(w^{ON}, f^{ON}) + T_1(w^{OFF}, f^{OFF}) = w^{ON}\frac{CPI^{ON}}{f^{ON}} + w^{OFF}\frac{CPI^{OFF}}{f^{OFF}} \quad (6).$$

Assuming ON-chip and OFF-chip frequencies are equal ($f^{ON} = f^{OFF}$), and $CPI = (CPI^{ON} + CPI^{OFF})/2$, this equation reduces to Equation 5. We observe that generally for ON-chip and OFF-chip workloads $f^{ON} \neq f^{OFF}$, meaning CPU and memory bus frequencies differ, and $CPI^{ON} \neq CPI^{OFF}$, meaning the workload throughput is different for ON- and OFF-chip workloads.

Parallel execution time on N processors for an ON-/OFF-chip workload with DOP=i ($T_N(w_i^{ON}, f^{ON})$,$T_N(w_i^{OFF}, f^{OFF})$)

[1] This does not account for out-of-order execution and overlap between memory access and computation, simplifying the discussion for now.

i: the *degree of parallelism* or DOP defined as the maximum number of processors that can be busy computing a workload for an observation period given an unbounded number of processors.

m: the maximum DOP for a given workload.

w_i: the amount of work (in instructions or computations) with *i* as the DOP.

w_i^{ON}: the number of ON-chip workloads with DOP=*i*.

w_i^{OFF}: the number of OFF-chip workloads with DOP=*i*.

N: the number of homogeneous processors available for computing the workloads.

w_{PO}: the parallel overhead workload due to extra work for communication, synchronization, etc.

$T(w_{PO},f)$: the execution time for parallel overhead w_{PO} for frequency *f*.

$T_N(w,f)$: the *parallel execution time* or the amount of time to complete workload *w* on *N* processors for frequency *f*.

The total amount of work (in instructions or computations) is given as $w = \sum_{i=1}^{m}\left(w_i^{ON} + w_i^{OFF}\right)$ where $1 \le i \le m$. Thus,

$$T_N(w_i,f) = T_N(w_i^{ON},f^{ON}) + T_N(w_i^{OFF},f^{OFF}) = \frac{w_i^{ON}}{i}\cdot\frac{CPI^{ON}}{f^{ON}} + \frac{w_i^{OFF}}{i}\cdot\frac{CPI^{OFF}}{f^{OFF}} \quad (7),$$

where $m \le N$.[2] Next, we include the additional execution time $T(w_{PO},f)$ for parallel overhead. We assume parallel overhead workload cannot be parallelized, but that it is divisible into ON-chip (w_{PO}^{ON}) and OFF-chip (w_{PO}^{OFF}) workloads. Thus

$$T_N(w,f) = \sum_{i=1}^{m}\left(T_N(w_i^{ON},f^{ON}) + T_N(w_i^{OFF},f^{OFF})\right) + T(w_{PO},f) \quad (8), \text{ and}$$

$$T_N(w,f) = \sum_{i=1}^{m}\left(\frac{w_i^{ON}}{i}\cdot\frac{CPI^{ON}}{f^{ON}} + \frac{w_i^{OFF}}{i}\cdot\frac{CPI^{OFF}}{f^{OFF}}\right) + \left(T(w_{PO}^{ON},f^{ON}) + T(w_{PO}^{OFF},f^{OFF})\right) \quad (9).$$

Power-aware speedup for DOP and ON-/OFF-chip workloads ($S_N(w,f)$)

f_0^{ON}: the lowest available ON-chip frequency.

$S_N(w,f)$: the ratio of sequential execution time ($T_1(w,f)$) to parallel execution time ($T_N(w,f)$).

On power-aware parallel systems, ON-chip frequency f^{ON} may change due to DVFS scheduling of the processor. As a consequence, power-aware speedup has two key variables: ON-chip clock frequency (f^{ON}) and the number of available processors (*N*) computing workload *w*. Speedup is computed relative to the sequential execution time to complete workload *w* on *1* processor at the lowest available ON-chip frequency, f_0^{ON}. Power-aware speedup is defined using Equations 6 and 9 as:

$$S_N(w,f) = \frac{T_1(w,f)}{T_N(w,f)} = \frac{w^{ON}\dfrac{CPI^{ON}}{f_0^{ON}} + w^{OFF}\dfrac{CPI^{OFF}}{f^{OFF}}}{\displaystyle\sum_{i=1}^{m}\left(\frac{w_i^{ON}}{i}\cdot\frac{CPI^{ON}}{f^{ON}} + \frac{w_i^{OFF}}{i}\cdot\frac{CPI^{OFF}}{f^{OFF}}\right) + \left(T(w_{PO}^{ON},f^{ON}) + T(w_{PO}^{OFF},f^{OFF})\right)} \quad (10).$$

Usage of power-aware speedup ($S_N(w,f)$)

Equation 10 illustrates how to calculate power-aware speedup. For a more intuitive description, assume the workload is broken into a serial portion (w_1) and a perfect parallelizable

[2] Strictly speaking, this limitation is not required. For $m > N$, we can add an $\lceil i/N \rceil$ term to Equation 5 and succeeding equations to limit achievable speedup to the number of available processors, *N*. We omit this term to simplify the discussion and resulting formulae.

portion (w_N) such that $w = w_1 + w_N$, $N=m$, and $w_i=0$ for $i \neq 1$, $i \neq m$. Then, allowing for flexibility in our execution time notation, we can express the power-aware speedup under these conditions as:

$$S_N(w,f) = \frac{T_1(w^{ON}, f_0^{ON}) + T_1(w^{OFF}, f^{OFF})}{\left[T_N(w_1^{ON}, f^{ON}) + T_N(w_1^{OFF}, f^{OFF})\right] + \left[T_N(w_N^{ON}, f^{ON}) + T_N(w_N^{OFF}, f^{OFF})\right] + \left(T(w_{PO}^{ON}, f^{ON}) + T(w_{PO}^{OFF}, f^{OFF})\right)} \quad (11).$$

Here, $T_1(w^{ON}, f_0^{ON}) + T_1(w^{OFF}, f^{OFF})$ is the base line sequential execution time unaffected by CPU frequency scaling or parallelism. $T_N(w_1^{ON}, f^{ON})$ is the sequential portion of the workload affected by CPU frequency scaling, but not affected by parallelism. $T_N(w_1^{OFF}, f^{OFF})$ is the sequential portion of the workload not affected by CPU frequency scaling or parallelism. $T_N(w_N^{ON}, f^{ON})$ is the parallelizable portion of the workload also affected by CPU frequency. $T_N(w_N^{OFF}, f^{OFF})$ is the parallelizable portion of the workload not affected by CPU frequency. $T(w_{PO}^{ON}, f^{ON})$ is the parallel overhead affected by CPU frequency. $T(w_{PO}^{OFF}, f^{OFF})$ is the parallel overhead not affected by CPU frequency.

4. POWER-AWARE PARALLEL SPEEDUP EVALUATION

In this section, we analyze the power-aware speedup for two classes of applications: computation-bound applications with negligible parallel overhead and communication-bound applications with significant parallel overhead. We use the embarrassingly parallel (EP) and Fourier transform (FT) benchmarks from the NAS Parallel Benchmark suite [2] for each category respectively. We note that our intention here is to show the accuracy of our approach for analytically quantifying the impact of power-aware features on execution time and speedup. We start with EP since the results are straightforward and as proof of concept for power-aware speedup. We describe results for more interesting codes, FT in the succeeding subsection.

4.1 Experimental Platform

The power-aware system used in these experiments is a 16-node DVS-enable cluster. It is constructed with 16 Dell Inspiron 8600s connected by a 100M Cisco System Catalyst 2950 switch. Each node is equipped with a 1.4 GHz Intel Pentium M processor using Centrino mobile technology to provide high-performance with reduced power consumption. The processor includes an on-die 32K L1 data cache, a on-die 1 MB L2 cache, and each node has 1 GB DDR SDRAM. Enhanced Intel Speedstep technology allows software to dynamically adjust the processor among five supply voltage and clock frequency settings given by Table 2.

Table 2. Operating points in frequency and supply voltage for the Pentium M 1.4GHz processor.

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

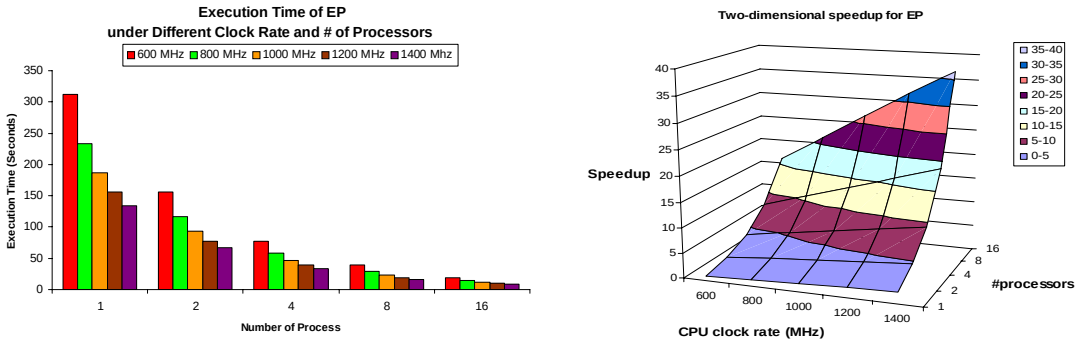
We installed open-source Linux Fedora Core and MPICH for communication on each node. We use DVS scheduling techniques similar to our previous work [15] on this system.

4.2 Power-aware Speedup for Computation-Bound Benchmark EP

Figure 1 shows the measured parallel execution time and power-aware speedup for the EP benchmark. EP evaluates an integral using a pseudorandom trial. Cluster-wide computations require virtually no inter-processor communication. The ratio of memory operations to computations on each node is very low. Figure 1 indicates the following for the EP workload:

1) Execution time (Figure 1a) for a fixed frequency can be reduced by increasing the number of nodes used in computations.

2) Execution time (Figure 1a) for a fixed processor count can be reduced by increasing the CPU clock rate.
3) Speedup (Figure 1b) for a fixed base frequency (600MHz) increases linearly with the number of processors. For instance, speedup increases from 1 at 1 processor to 15.9 at 16 processors.
4) Speedup (Figure 1b) for 1 processor increases linearly with processor frequency. For instance, speedup increases from 1.0 at 600MHz to 2.34 at 1400MHz.
5) The overall speedup using simultaneous enhancements of processor count and frequency is nearly the product of the individual speedups for each enhancement. For instance, the maximum speedup (36.5) measured on 16 processors for 1400MHz is almost equal to the product of measured parallel speedup (15.9) and frequency speedup (2.34).



1-a 1-b

Fig. 1. Figure (a) shows measured parallel execution time with varying clock rate; Figure (b) shows the speedup for scaled processor counts and frequencies.

We now use our power-aware speedup formulation to explain these observations analytically. A computation-bound application such as EP spends the majority of its execution time doing calculations on the CPU, and the time spent performing OFF-chip (i.e. memory) accesses is negligible. Thus, the workload w is only an ON-chip workload, or more formally $w = \sum_{i=1}^{m} w_i^{ON}$. OFF-chip workloads are negligible, or $\sum_{i=1}^{m} w_i^{OFF} = 0$. The characteristics of EP also indicate a majority of the workload can be completely parallelized, such that $w = \sum_{i=1}^{m} w_i = w_N$, where $N=m$, and $w_i=0$ for $i \neq m$. Hence, $w = \sum_{i=1}^{m} w_i^{ON} = w_N^{ON}$, and since EP exhibits almost no inter-processor communication, $w_{PO}^{ON} = w_{PO}^{OFF} = 0$. Under these assumptions, the analytical power-aware speedup for EP using Equation 11 is

$$S_N(w,f) = \frac{T_1(w,f_0)}{T_N(w,f)} = \frac{w_N^{ON}\dfrac{CPI^{ON}}{f_0^{ON}}}{\dfrac{w_N^{ON}}{N}\cdot\dfrac{CPI^{ON}}{f^{ON}}} = N \cdot \frac{f^{ON}}{f_0^{ON}} \qquad (12).$$

This application exhibits near perfect performance: easily parallelized workload, no overhead for communication, and nearly ideal memory behavior. Thus, the speedup predicted by Equation 12 is a simple product of the individual speedups for parallelism (N), and for frequency (f^{ON}/f_0^{ON}), where we are comparing a faster frequency (e.g. $f^{ON} = 1400$) to the base frequency ($f_0^{ON} = 600$). The predicted speedup for 16 processors (37.3) is within 2.3% of the measured speedup (36.5), and this error is the maximum error over all the predictions for EP.

Predicting the power-aware speedup for EP makes a reasonable case for using the product of individual speedups described by Equation 3 (Amdahl's Law generalization). The speedup of

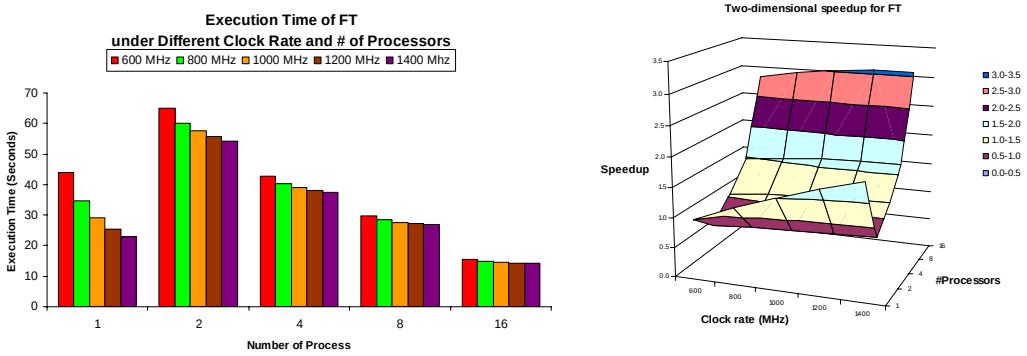
embarrassingly parallel (EP) applications with small memory footprints will always improve with increased processor count and frequency. Though EP prediction shows our methods are as accurate and useful as Amdahl's Law, this behavior is not typical of many parallel scientific applications such as FT. In the next subsections we use power-aware speedup techniques to analyze codes with significant parallel overhead (FT) and more complex memory behavior.

4.3 Power-aware Speedup for Communication-bound Benchmark FT

Figure 2 shows the measured parallel execution time and power-aware speedup for the FT benchmark. FT computes a 3-D partial differential equation solution using fast Fourier Transforms. Parallel FT iterates through four phases: computation phase 1, reduction phase, computation phase 2, and all-to-all communication phase. Both computation phases spend most of their time performing calculations but with a larger memory footprint than EP. The parallel overhead of the reduction and all-to-all communication phases dominate execution time. Figure 2 indicates the following for the FT workload:

1) Execution time (Figure 2a) for 2 or more processors is reduced by increasing the number of processors used in computations. However, the rate of improvement is sub-linear.
2) Execution time (Figure 2a) for 1 processor can be reduced by increasing CPU clock rate. However, the rate of improvement is sub-linear; from 1.0 at 600MHz to 1.9 at 1400MHz.
3) Speedup (Figure 2b) for a fixed base frequency (600 MHz) decreases from 1 to 2 processors. Speedup increases from 2 to 16 processors. For instance, for 600MHz speedup increases from 1.0 on 1 processor to 2.9 on 16 processors.
4) Speedup (Figure 2b) for 1 processor increases sub-linearly with processor frequency. For instance, speedup increases from 1.0 at 600MHz to 1.6 at 1400MHz.
5) The overall speedup using simultaneous enhancements of processor count and frequency is a complicated function. For example, the effects of frequency scaling on execution time (Figure 2a) diminish as the number of nodes increase.

We now use our power-aware speedup formulation to explain these observations analytically. First, we consider the workload run sequentially at various CPU clock frequencies. As mentioned, execution time decreases sub-linearly. This behavior differs from EP where the effects were linear. The memory behavior of FT, computing transforms on sizable matrices, takes more time and more OFF-chip accesses than EP. Thus, we cannot simplify the numerator of Equation 11 since we must consider both ON-chip and OFF-chip delays in the workload.



2-a **2**-b

Fig. 2. Execution time and two-dimensional speedup of FT by parallel computing and increasing CPU clock rate

Next, we consider the parallel overhead. A communication-bound application such as FT spends the majority of its execution time performing communications or parallel overhead, w_{PO}. From 1 to 2 processors, the execution time increases for all frequencies. This indicates the parallel overhead is significant and we must determine its effects on execution time. The parallel overhead for FT is actually dominated by all-to-all communications and synchronizations. In other work [5, 17], we observed such communication overhead is not affected significantly by CPU clock frequency. Thus, we claim $w_{PO}^{ON} = 0$, but w_{PO}^{OFF} is a significant portion of parallel execution time.

Table 3. To determine the best system configuration for FT for all combinations of frequency and number of nodes, we need pairwise speedup comparisons to the slowest frequency (600 MHz) and the smallest number of nodes (N=1) as the base sequential execution time. To predict speedup, we use Equation 13. Each table entry is the error or the difference between the measured and predicted speedup divided by the measured speedup. 600 MHz is used as the basis for comparison, so its column shows no error since it effectively varies only with number of nodes, exemplifying traditional speedup.

N	Frequency (MHz)				
	600	800	1000	1200	1400
2	0%	0.2%	0.2%	0.2%	0.3%
4	0%	0%	0.1%	0.1%	0.2%
8	0%	0.4%	2.0%	1.2%	0.7%
16	0%	2.1%	2.2%	2.3%	1.4%

Last, we consider the effects of parallelism. FT is not embarrassingly parallel, so the DOP can range from 1 to m. As the processor counts increase, execution time decreases. This indicates that a good portion of the workload is parallelizable and affected by m<=16. However, these effects lessen with the number of nodes and speedup tends to flatten out. In other work [10] on similar systems, we observed speedups for this workload do not change significantly from 16 to 32 nodes. Since our system exhibits similar behavior for this workload, $m=16=N$ seems a reasonable assumption – not to mention a limitation of our system[3]. Under these assumptions, the analytical power-aware speedup for FT using Equation 11 is

$$S_N(w,f) = \frac{T_1(w,f)}{T_N(w,f)} = \frac{w^{ON}\dfrac{CPI^{ON}}{f_0^{ON}} + w^{OFF}\dfrac{CPI^{OFF}}{f^{OFF}}}{\sum_{i=1}^{16}\left(\dfrac{w_i^{ON}}{i}\cdot\dfrac{CPI^{ON}}{f^{ON}} + \dfrac{w_i^{OFF}}{i}\cdot\dfrac{CPI^{OFF}}{f^{OFF}}\right) + T(w_{PO}^{OFF}, f^{OFF})} \quad (13).$$

This application exhibits less than perfect performance: workload with limited parallelization, significant overhead for communication, and time consuming memory behavior. Thus, the speedup predicted by Equation 13 is not a simple product of the individual speedups for parallelism and frequency as it was for EP. In fact, the errors in Table 3 reflect the errors incurred when using this simple product to predict the speedup.

Table 3 shows the errors for prediction of FT power-aware speedup using Equation 13. Here, the errors are reduced to a maximum of 3%. The power-aware speedup for FT captures all of the empirical observations we noted. For example, the diminishing effects of frequency scaling as number of nodes scale is due primarily to the increasing impact of parallel overhead ($T(w_{PO}^{OFF}, f^{OFF})$). For small numbers of nodes, the effects are lessened since the ON-chip workload

[3] Admittedly, it would be nice to confirm this result on a larger power-aware cluster. However, ours is one of only a few power-aware clusters in the US and there are few (if any) larger than 16 or 32 nodes. We are attempting to acquire a larger machine presently.

[5] Most speedup models are calculated only analytically. Thus, it is common to make the assumption that $w=w_1+w_N$. In practice, speedup analysis focuses solely on the parallelizable portion of the code and w_1 is considered negligible. We follow this common practice, though we are exploring ways to measure w_1 directly.

$\sum_{i=1}^{16}\left(\frac{w_i^{ON}}{i}\cdot\frac{CPI^{ON}}{f^{ON}}\right)$ makes up a large portion of total execution time. However, as the number of nodes increases, this portion decreases. With this decrease, parallel overhead eventually dominates. Thus the effects of frequency diminish since $w_{PO}^{ON}=0$.

5. POWER-AWARE SPEEDUP PARAMETERIZATION

Though we have shown that our power-aware speedup model is accurate, to this point we have purposely hidden the details of how to obtain our model parameters on real systems. We primarily use versions of Equations 10 and 11 to obtain speedup predictions. Equation 10 is our general power-aware speedup formula and Equation 11 is a simplified version. In this section, we show how to derive model parameters and apply them in both equations to predict power-aware speedup.

5.1 Power-aware speedup simplified parameterization

We begin using Equation 11 for our predictions. For this simplified parameterization, we make two assumptions.

Assumption 1: a majority of the workload can be completely parallelized, such that $w=\sum_{i=1}^{m} w_i = w_N$, where $N=m$, and $w_i=0$ for $i\neq m$. Under this assumption[5], sequential execution time is simplified as

$$T_1(w,f)=\left[T_1(w_N^{ON},f^{ON})+T_1(w_N^{OFF},f^{OFF})\right]=w_N^{ON}\cdot\frac{CPI^{ON}}{f^{ON}}+w_N^{OFF}\cdot\frac{CPI^{OFF}}{f^{OFF}} \tag{14}$$

parallel execution time is simplified as

$$\begin{aligned}T_N(w,f)&=\left[T_N(w_N^{ON},f^{ON})+T_N(w_N^{OFF},f^{OFF})\right]+\left(T(w_{PO}^{ON},f^{ON})+T(w_{PO}^{OFF},f^{OFF})\right)\\&=\frac{w_N^{ON}}{N}\cdot\frac{CPI^{ON}}{f^{ON}}+\frac{w_N^{OFF}}{N}\cdot\frac{CPI^{OFF}}{f^{OFF}}+\left(T(w_{PO}^{ON},f^{ON})+T(w_{PO}^{OFF},f^{OFF})\right)\\&=\frac{T_1(w,f)}{N}+\left(T(w_{PO}^{ON},f^{ON})+T(w_{PO}^{OFF},f^{OFF})\right)\end{aligned} \tag{15}$$

Assumption 2: parallel overhead is not affected by ON-chip frequency [29], i.e. $w_{PO}^{ON}=0$. Under Assumption 2, Equation 15 is reduces to

$$T_N(w,f)=\frac{T_1(w,f)}{N}+T(w_{PO}^{OFF},f^{OFF}) \tag{16}$$

Equation 16 holds for all the frequencies. Given the relationship shown in Equation 16, we now describe how to predict power-aware performance given a processor count and frequency.

Step 1. Measure the sequential and parallel execution time for each processor count running the workload at ON-chip base frequency, $T_N(w,f_0^{ON})=T_N(w_N,f_0^{ON})$.

Step 2. Derive the parallel overhead time using the measured times from Step 1 and Equation 16 such that $T_N(w_{PO}^{OFF},f^{OFF})$ is the parallel overhead $T(w_{PO}^{OFF},f^{OFF})$ for processor count N:

$$T_N(w_{PO}^{OFF},f^{OFF})=T_N(w,f_0^{ON})-\frac{T_1(w,f_0^{ON})}{N} \tag{17}.$$

Step 3. Measure the sequential execution time $T_1(w,f)=T_1(w_N,f)$ for each frequency running the workload on *1* processor.

Step 4. Use the derived parallel overhead in Step 2 and measured sequential execution time from Step 3 to predict the parallel execution time $T_N(w,f)$ for any given combination of processor count ($N>1$) and frequency ($f > f_0^{ON}$).

$$T_N(w,f) = \frac{T_1(w,f)}{N} + T_N(w_{PO}^{OFF}, f^{OFF}) = \frac{T_1(w,f)}{N} + \left[T_N(w, f_0^{ON}) - \frac{T_1(w, f_0^{ON})}{N} \right] \quad (18)$$

Tables **3** shows prediction errors for FT are less than 3% using this technique. With these results, our assumptions appear reasonable. In fact, this is a very practical means of obtaining power-aware speedup. Nonetheless, there are drawbacks to this approach. First, this technique requires measurements for the sequential ($T_1(w, f)$) and parallel ($T_N(w, f_0^{ON})$) execution time. Second, this technique does not separate ON-chip and OFF-chip workloads. Thus, the effects of frequency are accounted for but inseparable from the execution time. Third, the assumptions used are the root cause of observable error. Assuming perfect parallelism means over estimating the effects of increasing the number of processors. Assuming parallel overhead is not affected by frequency means under estimating the effects of increasing processor frequency. In the next section we eliminate the first two drawbacks and mitigate the third as much as possible.

5.2 Power-aware speedup fine-grain parameterization

In this section, we show how to derive and use detailed power-aware speedup parameters to predict performance. We use Equation 10 as the basis for our discussion. We have applied this technique to FT with error rates similar to those in Tables **3**. We use the lower-upper diagonal (LU) benchmark from the NAS Parallel Benchmark suite as a case study. LU uses a symmetric, successive overrelaxation numerical scheme to solve a regular-sparse, block lower and upper triangular system. LU is an iterative solver with a limited amount of parallelism and a memory footprint comparable to FFT. LU exhibits a regular communication pattern and makes use of the memory hierarchy.

This technique consists of three steps: workload distribution, workload time, and execution time prediction.

Table 5. Workload measurement and decomposition

Workload	Memory level	Derivation	#ins (×10^9)
ON-chip	CPU/Register	*PAPI_TOT_INS - PAPI_L1_DCA*	*145*
	L1 Cache	*PAPI_L1_DCA – PAPI_L1_DCM*	*175*
	L2 Cache	*PAPI_L2_TCA – PAPI_L2_TCM*	*4.71*
OFF-chip	Main Memory	*PAPI_L2_TCM*	*3.97*

Step 1: Workload distribution (w^{ON} , w^{OFF})

The goal for this step is to obtain the distribution of the ON-/OFF-chip workloads. On the system measured, ON-chip workloads consist of computations with data residing in the CPU, a register, and the L1 or L2 cache. OFF-chip workloads consist of computations with data residing in the memory or on a disk.

We use hardware counters to measure the workload parameters for LU. Hardware performance counters are special registers that accurately track low-level operations and events such as the number of executed instructions and cache misses with minimum overhead. Hardware limitations on the number and type of events counted simultaneously require us to run the application multiple times in order to record all the events we need. In this work, we use

PAPI [28] to interface with the counters. We assume hardware event counts are similar across different processors for the same workload and obtain measurements on 1 processor[6].

To quantify ON-chip workload distribution, we monitor the following PAPI events: total instructions (PAPI_TOT_INS), L1 data cache accesses (PAPI_L1_DCA), L1 data cache misses (PAPI_L1_DCM), L2 cache accesses (PAPI_L2_TCA), and L2 cache misses (PAPI_L2_TCM).

Table 5 shows the formulae[7]. and the workload distributions for LU in number of instructions and percentage of ON-chip workload. ON-chip workloads (w^{ON}) account for 98.8% of the total workload. Despite LU's significant memory footprint, most data (97.4%) is available in the L1 cache. OFF-chip workloads (w^{OFF}) account for 1.2% of the total workload. We can also determine the distribution of the ON-chip workload: 44.66% CPU/Register instructions, 53.89% L1 cache instructions, 1.45% L2 cache instructions. The OFF-chip workload consists of only memory instructions.

Table 6. Seconds per Instruction (*CPI/f*) for ON-/OFF-chip workload

		600MHz	800MHz	1000MHz	1200MHz	1400MHz
w^{ON}	*ON-chip* CPI^{ON}	2.19	2.19	2.19	2.19	2.19
	CPI^{ON}/f^{ON} $(\times 10^{-9})$	3.65	2.74	2.19	1.83	1.56
w^{OFF}	CPI^{OFF}/f^{OFF} $(\times 10^{-9})$	140	140	110	110	110
w_{PO}	*For 155 doubles* $(\times 10^{-6})$	25	25	25	25	25
	For 310 doubles $(\times 10^{-6})$	200	167	167	167	167

Step 2: Workload time (CPI^{ON}/f^{ON}, CPI^{OFF}/f^{OFF}, and $T_N(w_{PO}, f)$)

Next, we measure the average amount of time (CPI_j/f) required for each of the four types of workload identified in the previous step (where j=[1,2,3,4]=[CPU/Register, L1 cache, L2 cache, memory]). We use the LMBENCH [26] toolset as it enables us to isolate the latency for each of these workload types. Using the weighted ON-chip workload distribution identified in the previous step, we can calculate the weighted average *CPI/f* for ON-chip workloads, $CPI^{ON}/f = .446CPI_1/f + .538CPI_2/f + .014CPI_3/f$ where f is any of the available frequencies[9]. Similarly, the weighted average *CPI/f* for OFF-chip workloads is $CPI^{OFF}/f = CPI_4/f$.

Table 6 presents the seconds per workload for ON-/OFF-chip workloads for each available processor frequency. Our premise is that ON-chip workloads are affected by frequency while OFF-chip workloads are not. The results in Table 6 show CPI^{ON}/f^{ON} or seconds per ON-chip workload decrease with frequency scaling. Table 6 shows OFF-chip workloads are basically constant with frequency scaling. On our system, we measured a slight increase in seconds per memory workload for slower CPU clock frequencies. We believe this is due to a hardware-driven decrease in the bus speed (f^{OFF}) for lower CPU clock frequencies. This is system-specific

[6] This technique is commonly used for regular SPMD codes such as LU. We observe the performance event counts are within 2% from sequential execution to parallel execution. For non SPMD codes, we could obtain results from individual processors and perform similar (albeit more cumbersome) analyses.

[7] We use event count of data cache access to approximate total cache access due to limited available event count from performance counter..

[9] This assumes one floating point double (FPD) computation per memory operation. For the actual predictions, we adjust CPI^{ON} to account for instruction-level parallelism that enables about 2.42 FPD computations per memory operation.

behavior captured by our parameter measurements. So, the effects are included in our predictions. Nonetheless, we are investigating this further to determine if it is common across platforms.

To measure communication workload time ($T_N(w_{PO}, f)$), we measure the seconds per communication for different message sizes using the MPPTEST [19] toolset. We observe that LU transmits 310 doubles per message between two nodes. For four nodes, LU transmits 155 doubles per message. Table 6 shows the transmission times for each of these cases. The trend is similar as the number of nodes increases. For the larger message sizes (310 doubles) on the slowest frequency, the communication time is influenced by the CPU (f^{ON}). For smaller message sizes on more than 2 nodes, CPU frequency has no noticeable effects. We use the product of number of messages and message time to compute $T_N(w_{PO}, f)$.

Step 3. Execution time and speedup prediction

Now, we can predict the execution time of LU for combinations of processor count and frequency using Equations 14 and 15 and the parameter values from Steps 1 and 2. We use Equation 14 to predict sequential execution time, $T_1(w, f)$. This means we rely on Assumption 1, that the total workload is parallelizable. We use Equation 15 to predict parallel execution time, $T_N(w, f)$, where we use $T_N(w_{PO}, f)$ from the previous step for the number of messages obtained by profiling LU.

Table 7. Power-aware speedup errors for LU. FP uses fine-grain parameterization to perform predictions. SP uses simplified parameterization to perform predictions.

N	Frequency (MHz)									
	600		800		1000		1200		1400	
	FP	SP	FP	SP	FP	SP	FP	SP	FP	SP
1	5%	0%	7%	0%	3%	0%	4%	0%	1%	0%
2	6%	0%	6%	2%	5%	4%	6%	3%	8%	6%
4	2%	0%	6%	3%	8%	4%	10%	7%	7%	7%
8	3%	0%	1%	4%	8%	8%	11%	10%	7%	13%

Table 7 presents the prediction error of this fine-grain parameterization (FP) and a comparison with simplified parameterization (SP). From this table, we observe that the errors for SP parameterization increase steadily with both number of nodes and frequency. Errors for FP increase with number of nodes but appear to be leveling off with frequency. Our assumptions explain these observations. Assuming the workload is completely parallelizable in both techniques increases the error rates. We are working presently to obtain better estimates of DOP to help mitigate these errors – though all speedup models suffer this problem. In the FP case, we separate the ON- and OFF-chip workloads. Thus, we are able to improve the insight and accuracy of the SP method. Of course, FP requires additional parameterization studies.

6. RELATED WORK

The related work of this paper focuses on two aspects: studies on scalability and speedup models for parallel applications, and studies on performance and energy tradeoffs in power-aware high performance computing.

Several speedup models have been proposed for parallel applications in high-performance computing[1, 9, 18, 20, 25, 30-32]. The first model is Amdahl's law [1]. Amdahl's Law states that speedup is limited by the fraction of the workload that can be computed in parallel. It addresses the speedup for fixed problem size, and it is also called strong scaling. Gustafson

proposed a fixed time speedup model for scaled problem size [20]. It quantifies the ability to scale up the workload to improve accuracy within the same execution time when more computational nodes are available. Sun et al have argued that the problem size can be scaled further in large memory systems to gain more speedup and improved accuracy [30]. They presented a speedup model under which the scaled workload is constrained by memory. While fixed-time speedup and memory-bounded speedup address scaling workload to gain speedup and accuracy, the isoefficiency metric proposed by Grama et al [18] studies how to keep the same efficiency by scaling the workload.

Studies on power-aware high performance computing are mainly focusing on reducing energy consumption without affecting performance. Hsu and Kremer studied compiler directed dynamic voltage and frequency scheduling for memory-bounded sequential workloads [21]. Cameron et al have demonstrated significant energy saving could be achieved with minimum performance impact by variable DVS scheduling in parallel scientific computing [4, 6, 11, 14, 16]. Freeh et al studied potential energy savings in parallel MPI applications using the NPB benchmarks[12, 13, 29]. Chen et al suggested scaling down the CPU speed on nodes that are not in the critical path to save energy without performance penalty [7, 24].

7. CONCLUSIONS

In this paper, we present a power-aware speedup model for emergent power-aware distributed systems. By decomposing the workload with DOP and ON-/OFF-chip characteristics, this model takes into account the effects of both parallelism and power aware techniques on speedup. Our study of several NPB codes on a DVS-enabled power aware cluster shows that our power-aware model is able to capture application characteristics and their effects on speedup. Coupled with an energy-delay metric, this new speedup model can predict both the performance and the energy/power consumption.

In the future, we will further refine our work by applying this power-aware model to code segment granularity according to the workload characteristics inside parallel applications.

8. REFERENCES

[1] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," Proceedings of AFIPS Spring Joint Computer Conference, Reston, VA, 1967.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The Nas Parallel Benchmarks," *International Journal of Supercomputer Applications and High Performance Computing*, vol. 5, pp. 63-73, 1991.

[3] D. Brooks, M. Martonosi, J.-D. Wellman, and P. Bose, "Power-Performance Modeling and Tradeoff Analysis for a High End Microprocessor," Proceedings of Workshop on Power-Aware Computer Systems (PACS2000, held in conjuction with ASPLOS-IX), Cambridge, MA, 2000.

[4] K. W. Cameron, R. Ge, and X. Feng, "High-Performance, Power-Aware Distributed Computing for Scientific Applications," *IEEE Computer*, vol. 38, 2005.

[5] K. W. Cameron, R. Ge, and X. Feng, "High-Performance, Power-Aware Distributed Computing for Scientific Applications," *Computer*, vol. 38, pp. 40-47, 2005.

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

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

[8] K. Choi, R. Soma, and M. Pedram, "Dynamic Voltage and Frequency Scaling based on Workload Decomposition," Proceedings of the 2004 international symposium on Low power electronics and design, Newport Beach, California, USA, 2004.

[9] D. Eager, J. Zahorjan, and E. Lazowska, "Speedup versus efficiency in parallel system," *IEEE Trans. Comput.*, pp. 403-423, March 1989.

[10] X. Feng, R. Ge, and K. W. Cameron, "The Argus Prototype: Aggregate Use of Load Modules as a High density Supercomputer," *Concurrency and Computation: Practice and Experience*, vol. 18, 2006.
[11] X. Feng, Rong Ge, Kirk Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," Proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS 05), Denver, CO, 2005.
[12] V. W. Freeh, D. K. Lowenthal, F. Pan, and N. Kappiah, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," Proceedings of 10th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP), 2005.
[13] V. W. Freeh, D. K. Lowenthal, R. Springer, F. Pan, and N. Kappiah, "Exploring the Energy-Time Tradeoff in MPI Programs," Proceedings of 19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS), Denver, Colorado, 2005.
[14] R. Ge, X. Feng, and K. Cameron, "Performance- and Energy-Conscious Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," Proceedings of Proceedings of 16th International Conference on High Performance Computing and Communications (SC 2005), Seatle, WA, 2005.
[15] R. Ge, X. Feng, and K. Cameron, "Performance-constrained, Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," Proceedings of 2005 ACM/IEEE conference on Supercomputing (SC 2005), Seattle, WA, 2005.
[16] R. Ge, X. Feng, and K. W. Cameron, "Improvement of Power-Performance Efficiency for High-End Computing," Proceedings of 19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS), Denver, Colorado, 2005.
[17] R. Ge, X. Feng, and K. W. Cameron, "Improvement of Power-Performance Efficiency for High-End Computing," Proceedings of the first HPPAC workship in conjection with 19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS), Denver, Colorado, 2005.
[18] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: measuring the scalability of parallel algorithms and architectures," *IEEE Parallel & Distributed Technology: Systems & Technology* vol. 1, pp. 12-21.
[19] W. Gropp and E. Lusk, "Reproducible Measurements of MPI Performance," Proceedings of PVM/MPI '99 User's Group Meeting, 1999.
[20] J. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol. 31, pp. 532-533, 1988.
[21] C.-H. Hsu and U. Kremer, "Compiler-directed dynamic voltage scaling for memory-bound applications," Department of Computer Science Rutgers University, Piscataway August 2002.
[22] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," Proceedings of ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI'03), San Diego, CA, 2003.
[23] C.-H. Hsu and U. Kremer, "Dynamic Voltage and Frequency Scaling for Scientific Application," Department of Computer Science, Rutgers University DCS-TR447, 2001.
[24] N. Kappiah, V. W. Freeh, D. K. Lowenthal, and F. Pan, "Exploiting Slack Time in Power-Aware, High-Performance Programs," Proceedings of IEEE/ACM Supercomputing 2005 (SC|05), Seattle, WA, November, 2005.
[25] A. H. Karp and H. P. Flatt, "Measuring parallel processor performance," Proceedings of CACM, May 1990.
[26] L. McVoy and C. Staelin, "lmbench: Portable tools for performance analysis," Proceedings of USENIX 1996 Annual Technical Conference, San Diego, CA, 1996.
[27] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A quantitative approach*, 3rd ed. San Fancisco, CA: Morgan Kaufmann Publishers, 2003.
[28] PTOOLS, "Performance API Home Page," 1999. "icl.cs.utk.edu/projects/papi"
[29] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh., "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," Proceedings of 11th ACM Symposium on Principles and Practice of Parallel Programming (PPOPP), 2006.
[30] X.-H. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *Journal of Parallel and Distributed Computing*, vol. 19, pp. 27-37, 1993.
[31] X. H. Sun and L. M. Ni, "Another view on parallel speedup," Proceedings of Supercomputing '90, New York, 1990.
[32] P. H. Worley, "The effect of time constraints on scaled speedup," *SIAM J. Sci. Stat. Comput*, vol. 11,5, pp. 838-858, 1990.
[33] Q. Wu, V. J. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," Proceedings of the 38th IEEE/ACM International Symposium on Microarchitecture (MICRO-38), Barcelona, Spain, 2005.