Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores

Matthew Curtis-Maury, Ankur Shah,
Filip Blagojevic, Dimitrios S. Nikolopoulos
Dept. of Computer Science, Virginia Tech
Blacksburg, VA, USA
mfcurt@cs.vt.edu, ankur77@cs.vt.edu,
filip@cs.vt.edu, dsn@cs.vt.edu

Bronis R. de Supinski, Martin Schulz
Lawrence Livermore National Laboratory
Livermore, CA, USA
bronis@llnl.gov, schulzm@llnl.gov

ABSTRACT

Power has become a primary concern for HPC systems. Dynamic voltage and frequency scaling (DVFS) and dynamic concurrency throttling (DCT) are two software tools (or *knobs*) for reducing the dynamic power consumption of HPC systems. To date, few works have considered the synergistic integration of DVFS and DCT in performance-constrained systems, and, to the best of our knowledge, no prior research has developed application-aware simultaneous DVFS and DCT controllers in real systems and parallel programming frameworks. We present a multi-dimensional, online performance predictor, which we deploy to address the problem of simultaneous runtime optimization of DVFS and DCT on multi-core systems. We present results from an implementation of the predictor in a runtime library linked to the Intel OpenMP environment and running on an actual dual-processor quad-core system. We show that our predictor derives near-optimal settings of the power-aware program adaptation knobs that we consider. Our overall framework achieves significant reductions in energy (19% mean) and ED^2 (40% mean), through *simultaneous* power savings (6% mean) and performance improvements (14% mean). We also find that our framework outperforms earlier solutions that adapt only DVFS *or* DCT, as well as one that sequentially applies DCT *then* DVFS. Further, our results indicate that prediction-based schemes for runtime adaptation compare favorably and typically improve upon heuristic search-based approaches in both performance and energy savings.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: [Parallel Architectures]; D.4.1 [**Operating Systems**]: Process Management—*Concurrency*; D.4.8 [**Operating Systems**]: Performance—*Modeling and prediction*

General Terms

Management, Measurement, Performance

PACT'08, October 25–29, 2008, Toronto, Ontario, Canada.

1. INTRODUCTION

Multi-core processors can trade parallelism and performance for reduced power consumption through software control of the number of active cores and power-aware workload distribution between cores [18]. In workload execution phases with limited scalability, controlling concurrency and workload distribution produces substantial energy savings with no performance penalty. Occasionally, throttling concurrency provides a performance gain by reducing contention between threads for shared resources such as memory bandwidth. Conserving cores at runtime can be a valuable optimization for emerging processors with hundreds of cores [1]. Recent studies indicate that less than half of industrial-strength parallel codes scale to hundreds of conventional single-core processors and only a handful scale to thousands of processors [13]. Thus, conserving cores, either for power saving purposes, or for other purposes, such as consolidation or fault tolerance [15], are viable alternatives to unconstrained parallelization.

Dynamic concurrency throttling (DCT), which adapts the level of concurrency at runtime based on execution properties, is a software-controlled mechanism, or *knob*, for runtime power-performance adaptation on systems with multi-core processors. Dynamic voltage and frequency scaling (DVFS) provides a second knob. While earlier research has significantly advanced the understanding of performance and power implications of DVFS [11, 19] and DCT [4], less emphasis has been placed on integrating DVFS and DCT in a unified power-performance adaptation framework. In particular, these techniques have not yet been applied in synergy to real HPC systems and applications, neither have they been considered in the context of parallel programming models and runtime environments.

Combined approaches for simultaneous DVFS and DCT in parallel applications have been explored recently via hardware simulation, using empirical search algorithms [18]. This study clearly demonstrated that runtime adaptation of concurrency and voltage/frequency in iterative parallel applications can achieve high-performance, power-efficient execution. Unfortunately, even with only two power-performance adaptation knobs available in software, the search space for adaptation can grow to unmanageable proportions. An M-core processor with L voltage/frequency levels presents a power-performance adaptation search space of size $O(L \cdot M)$. If we assume that for a given degree of concurrency, performance is sensitive to the placement of threads on cores, for example due to asymmetry in the core architecture or asymmetry in resource sharing between groups of cores, the space grows to $O(L \cdot 2^M)$. We cannot reasonably search this space, even with a modest number of cores and power states, particularly at runtime.

This paper presents a framework for multi-dimensional, online, software-controlled, and HPC-constrained power-performance adaptation on systems based on multi-core processors. The framework provides transparent runtime adaptation of HPC codes and requires only a trivial code instrumentation step. Its key component, a dynamic multi-dimensional performance predictor, statistically analyzes samples of hardware event rates collected from performance monitors and predicts the performance impact of any thread mapping and any DCT and DVFS levels available on the system (either combined or in isolation). We base the statistical analysis on a rigorous regression model that is trained from samples of the power-performance adaptation search space collected from real workloads. We derive a phase-aware performance prediction model with low runtime overhead, which dynamically adjusts DCT, DVFS, and thread placement at the granularity of program phases, regardless of input. The model usually predicts the optimal system configuration to execute each phase—occasional small errors lead it to choose a near optimal configuration instead—thereby achieving performance gains and energy savings.

We present a functioning software prototype of our framework linked with the Intel OpenMP runtime environment and evaluate it through physical experimentation on a system with two quad-core Intel Xeon processors. The model derivation and training are automated and portable across multi-core processors. Further, the associated software prototype is based on portable components, specifically PAPI and OpenMP.

Through derivation and evaluation of our prototype, this paper contributes answers to several important questions:

- Can statistical performance models based on hardware event counters accurately predict performance with multi-dimensional input parameters—more specifically, the number of cores, mapping of threads to cores, and processor voltage/frequency—across a large space of untested system configurations?
- Can prediction-based models achieve as good or better results than heuristic search methods that time the phases of the program on sample configurations for multi-dimensional power-performance adaptation?
- Do prediction-based model costs prevent their use for online power-performance adaptation, given multiple knobs?
- Can we prune the optimization space for prediction-based power-performance adaptation during model training and during model actuation to derive effective adaptation frameworks without prohibitive development and training costs?
- Which power-performance adaptation knob—DCT or DVFS—is more critical with respect to power-efficiency, assuming no tolerance for performance loss in an HPC environment?
- What are the synergistic effects of applying these knobs simultaneously, if any?

Our experimental results demonstrate that simultaneous phase-aware prediction of the performance impact of DVFS and DCT can achieve significant energy and energy-delay^2 (ED^2) savings (19% mean and 40% mean respectively for the NAS benchmarks). In nearly all applications we tested, the energy gains come with simultaneous power savings (6% mean) and performance improvement (14% mean). Since our prediction schemes converge rapidly to optimal or near-optimal system configurations for parallel execution phases, they typically outperform exhaustive or heuristic search strategies and scale better than search strategies to many cores. We further show experimentally that a unified 2-dimensional DVFS-DCT predictor achieves both higher energy savings and performance gain, compared to a predictor that first predicts DCT and then DVFS, or either in isolation. We also show that our framework achieves substantial performance and energy gains over a transparent, OS-level DVFS tool based on ACPI. We conclude that users in performance-sensitive settings should apply the unified model of DVFS and DCT to sustain high performance and to obtain near maximum energy savings. Last, our results show that prediction-based power-performance adaptation schemes come very close to optimal static execution schemes, which can be derived only post-facto, after exhaustive experimentation.

The rest of this paper is organized as follows. Section 2 presents background and preliminary concepts. Section 3 presents our model of multi-dimensional power-performance prediction. Section 4 outlines the implementation of our prediction model in a real software prototype. Section 5 presents our experimental analysis from a system with two quad-core Intel Xeon processors. Section 6 discusses related work and Section 7 concludes the paper.

2. PRELIMINARIES

We provide the theoretical foundation for our performance predictor for systems with multiple multi-core processors. Since different mappings of a set of threads to cores may yield significant performance variation, we differentiate the number and the topology of the cores on each processor, as well as the number and topology of the processors during performance prediction. For example, on a system with multiple Intel Xeon quad-core processors, we differentiate between pairs of cores that share the L2 cache on the same processor, pairs of cores that do not share the L2 cache on the same processor, and pairs of cores that lie on different processors.

We assume that we can set each processor of the system to execute at an independent voltage/frequency level chosen from a predetermined set by a privileged instruction. We assume global voltage and frequency scaling for each processor as a whole, as opposed to per core, since this technology is readily available on commercially available multi-core processors. We conduct physical experimentation, using hardware timers and power meters to measure performance and energy respectively. Our modeling methodology does not preclude and can be generalized to local (per-core) DVFS schemes.

We decompose parallel workloads into phases, where each phase executes parallel computation using a potentially variable number of threads and completes at a synchronization point, such as a barrier, or a critical section. While DVFS is entirely transparent and can be applied to any code region, we cannot apply DCT to arbitrary parallel code regions without violating correctness. In principle, codes written in a shared-memory model where parallel computation is independent of the number of threads, are amenable to DCT without correctness considerations. The vast majority of OpenMP codes meet this requirement for processor-independence, as do the workloads that we use in this paper (NAS benchmarks). Other ongoing research efforts are addressing DCT in other programming models, such as MPI [10].

Our contribution involves a modeling/prediction component and a runtime actuation component. The first component predicts performance for each phase of parallel code under all feasible concurrency configurations and global voltage/frequency settings, with input from samples of hardware event counters collected at runtime. We use it at the boundaries of execution phases as the program executes. The predictor correlates hardware event counter samples, concurrency configurations (number and mapping of threads to cores), and voltage/frequency settings with whole system instruction throughput. Without loss of generality, we derive predictions for the fixed optimality criterion of minimizing energy without increasing runtime, which best meets the requirements of HPC environments. We use our predictions in the actuation component on

a per phase basis to minimize energy consumption under this rigid performance constraint.

3. EMPIRICAL MODEL DERIVATION

We present runtime performance predictors that estimate performance in response to changing the settings of two power-performance knobs, DCT and DVFS, where we differentiate between alternative available topologies for mapping threads to cores at any given DCT level. We refer to each combination of frequency and concurrency configuration available on the system as a hardware configuration, or more simply a *configuration*. The predictors use input from execution samples collected at runtime on specific configurations to predict the performance on other, untested configurations. We estimate performance for each phase in terms of useful instructions per second, or $uIPC$, which is the IPC with instructions used for parallelization or synchronization omitted. By using $uIPC$ predictions, we exploit opportunities to save power primarily by scaling the memory-bound parts of the actual computation to reduce contention and to exploit slack due to memory or parallelization stalls. The input from the sample configurations consists of the useful IPC ($uIPC_s$), as well as a set of n hardware event rates ($e_{(1..n,s)}$) observed for the particular phase on the sample configuration s, where each event rate $e_{(i,s)}$ is calculated as the number of occurrences of event i divided by the number of elapsed cycles during the execution of configuration s. The model predicts $uIPC$ on a given target configuration t, which we denote by $uIPC_t$.

3.1 Baseline Prediction Model

Our predictor model uses $uIPC_s$ to estimate the effect of the observed event rates that produce the resulting value of $uIPC_t$. The event rates capture the utilization of particular hardware resources that represent scalability bottlenecks, thereby providing insight into the likely impact of hardware utilization and contention on scalability. Although the model can include multiple sample configurations, we begin by describing the simplest case of a single sample and build up the model from there. We model $uIPC_t$ scalability as a linear function as follows:

$$uIPC_t = uIPC_s \cdot \alpha_t(e_{(1..n,s)}) + \epsilon_t \quad (1)$$

Equation 1, reflects the dependence of the function α_t and the constant term ϵ_t on the particular target configuration. That is, we model each target configuration t through coefficients that capture the varying effects of hardware utilization at different degrees of concurrency, different mappings of threads to cores, or different voltage/frequency levels. In effect, $\alpha()$ scales up or down the observed $uIPC_s$ on the sample configuration based on the observed values of the event rates on the same configuration, to attain $uIPC_t$ on any of the target configurations. The observed event rates determine how much we scale $uIPC_s$ as a linear combination of the sample configuration event rates as depicted in Equation 2:

$$\alpha_t(e_{(1..n,s)}) = \sum_{i=1}^{n}(x_{(t,i)} \cdot e_{(i,s)} + y_{(t,i)}) + z_t \quad (2)$$

The model's intuition is that changes in event rates indicate varying resource utilization and contention, resulting in either positive or negative effects on $uIPC_s$, which the model represents through positive or negative coefficients. While the relationship between event rates and $uIPC$ may not be strictly linear, a linear model can represent this relationship well [6, 14, 22]. We estimate the specific coefficients through multivariate linear regression as discussed further in Section 3.3. By using an empirical model, we greatly simplify the retraining process required for new architectures since we automatically infer the model from a set of training samples rather than through a detailed architectural description. We combine Equations 1 and 2, to derive the following equation for $uIPC$ on a particular target configuration t using a single sample configuration as:

$$uIPC_t = uIPC_s \cdot \sum_{i=1}^{n}(x_{(i,t)} \cdot e_{(i,s)}) + uIPC_s \cdot \gamma_t + \epsilon_t \quad (3)$$

Therefore, estimating the value of $uIPC_t$ is equivalent to the proper approximation of the coefficients $x_{(i,t)}$, the constant term ϵ_t, and γ_t. The γ_t variable is the sum of a collection of terms from α_t that represent a coefficient for $uIPC_s$ itself, independent of the values of ($e_{(1..n,s)}$), and defined as $\sum_{i=1}^{n}(y_{(t,i)}) + z_t$.

3.2 Model Extensions

While the baseline prediction model can be effective for DCT [5, 6], we refine it to improve model accuracy and extend the model to predict performance with multi-dimensional input. Our first extension models $uIPC_t$ as a linear combination of multiple sample configurations from the configuration space. In the context of DVFS and DCT, each sample configuration uses a different number of threads bound to different execution units in the machine, at potentially different voltage and frequency levels. Thus, each sample configuration provides some additional insight into execution on other, untested configurations. The use of multiple samples allows the model to "learn" more about each program phase's execution properties that determine performance on alternative configurations. The actual selection of the samples can be statistical (e.g., uniform), or empirical, i.e., using some architectural insight such as the number of cores sharing an L2 cache on each socket. Equation 4 presents the model extended to two samples, with an additional term λ to capture interaction between samples which we describe next.

$$\begin{aligned} uIPC_t = uIPC_{s1} \cdot \alpha_{(t,s1)}(e_{(1..n,s1)}) + \\ uIPC_{s2} \cdot \alpha_{(t,s2)}(e_{(1..n,s2)}) + \\ \lambda_t(e_{(1..n,S)}) + \epsilon_t \end{aligned} \quad (4)$$

Using multiple samples allows us to analyze the relationship between each configuration. We include an interaction term for the product of two events in the linear model to capture the relationship statistically. For simplicity, we only consider possible interactions between the same event across multiple configurations, including the product of $uIPC$ on each sample configuration. Thus, our model considers the interplay between multiple configurations. Specifically, we define the interaction term for a model using two samples as Equation 5 shows:

$$\begin{aligned} \lambda_t(1..n, S) = \sum_{i=1}^{n}(\mu_{(t,i)} \cdot e_{(i,s1)} \cdot e_{(i,s2)}) + \\ \mu_{(t,IPC)} \cdot uIPC_{s1} \cdot uIPC_{s2} + \iota_t \end{aligned} \quad (5)$$

The interaction term λ_t linearly combines the products of each event across configurations, as well as that of $uIPC$. In Equation 5, μ is the target-configuration-specific coefficient for each event pair and ι is the event rate independent term in the model.

On architectures with very large, complex configuration spaces, we may need to use even more sample configurations. We can extend our model to an arbitrary collection of samples, S, of size $|S|$, to support such a situation, as follows:

$$uIPC_t = \sum_{i=1}^{|S|}(uIPC_i \cdot \alpha_{(t,i)}(e_{(1..n,i)})) + \lambda_t(e_{(1..n,S)}) + \epsilon_t \quad (6)$$

While using more samples generally increases model accuracy, it also increases sampling overhead. We address the selection of S in terms of specific configurations as well as its size in Section 3.6.

We generalize the term λ_t further to account for the interaction between events across $|S|$ samples as follows:

$$\lambda_t(e_{(1..n,S)}) = \sum_{i=1}^{n}(\sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,i,j,k)} \cdot e_{(i,j)} \cdot e_{(i,k)}))) + \sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,j,k,IPC)} \cdot uIPC_j \cdot uIPC_k)) + \iota_t \quad (7)$$

To further improve model accuracy, we apply variance stabilization in the form of a square-root transformation to the data to reduce the correlation between the residuals and the fitted values, as is done by Lee, et al. [16]. That is, we take the square-root of each term, as well as the response variable, before applying the model. This process results in a more accurate model by reducing model error for the largest and smallest fitted values and by causing residuals to follow a normal distribution more closely.

3.3 Offline Model Training

We use multivariate linear regression on phases from a set of training benchmarks to approximate the coefficients in our model. We record the $uIPC$ and a predefined collection of event rates while executing each training benchmark's phases on all configurations. We use multiple linear regression on these values to learn the patterns in the effects of sample configuration event rates on the resulting $uIPC$ on the target configuration, with each phase's data serving as a training point. Specifically, the $uIPC$, the product of IPC and each event rate, and the interaction terms on the sample configurations serve as *independent variables* and the $uIPC$ on each target configuration serves as the *dependent variable*, in accordance with the above equations. We develop a model separately for each target configuration, deriving sets of coefficients independently. We select training benchmarks empirically, to include variation in properties such as scalability and memory-boundedness.

Testing all sample and target configurations offline for training purposes may become a time consuming process on architectures with many processing elements and/or many layers of parallelism. To combat this, we prune the target configuration space, using insight on the target system architecture. Specifically, we eliminate symmetric cases in thread binding as well as unbalanced bindings of threads. We also assume that the voltage/frequency of all dies in the system is set simultaneously to the same setting, to better support parallel codes and avoid load imbalance during parallel execution phases. On processors that feature hundreds of cores, it may become necessary to further reduce the search space during model training to limit offline overhead, for example by uniform sampling of the system configurations used for training. At current multi-core system scales, the training process using a fully automated system for our approach takes approximately five hours and scales up linearly with the number of possible configurations.

3.4 Event Selection Process

The model requires feedback from hardware event rates in order to predict performance across configurations accurately. Thus, we must identify particular event rates that result in high prediction accuracy. Unfortunately, the particular events that most reflect the performance impact of power knob settings are not always obvious. To select the events to use with our model, we use correlation analysis to determine which event rates on the sample configuration are most strongly correlated with the target IPCs. Then we select the top n events from the sorted list. We determine the number of events to use, n, based on how many event registers are available on the target architecture. The event selection process is statistical and automated, therefore portable across multi-core processors.

3.5 Predicting Across Multiple Dimensions

We can apply our model to predict the performance effects of DCT and DVFS independently, or across simultaneous changes in the settings of both power knobs. To predict for simultaneous changes, we collect samples at points along the two-dimensional space by varying the configuration along each prediction dimension. While we could predict along one dimension at a time by selecting the optimal configuration in each dimension sequentially, predicting along both dimensions simultaneously avoids blindspots in the predictions. The former strategy only predicts along the second dimension at the decided optimal level of the first dimension, whereas the second strategy is more likely to find the globally optimal configuration along both dimensions since it considers all combinations of both dimensions. We can generalize the model to predict performance in a configuration space of higher dimensions, and we can prune the space through uniform or other sampling schemes to reduce model training overhead.

3.6 Selecting Sample Configurations

Although we could randomly sample the configurations to reduce training and runtime search overhead, intuitively some configurations reveal specific architectural bottlenecks to scalability and performance. That is, certain configurations provide further insight into utilization of shared caches and memory bandwidth, and, thus, are stronger predictors than others. We therefore consider architectural properties while selecting the configurations that will best serve the prediction model.

When predicting along a single dimension (i.e., concurrency or DVFS-level), we use a single sample configuration at the maximum concurrency and frequency available. When predicting along two dimensions, in addition to sampling at maximum concurrency and frequency, we draw two more samples from points where concurrency is halved—with different mappings of the halved concurrency to cores—and frequency is reduced to the next lower available setting. This technique allows us to limit the number of samples, while providing input for the predictor along each dimension.

4. IMPLEMENTATION

We have implemented our multi-dimensional prediction model within a runtime library to perform online adaptation of DVFS and DCT. We target parallel applications from the HPC domain with an iterative structure, such that each program phase is executed many times. We exploit this property to collect hardware event rates during the first few executions of each phase to serve as input for the model. We hardcode the model itself into the runtime system by programming the coefficients derived during the training process for a particular model into the library. The runtime system facilitates online predictions of performance based on the collected hardware event rates.

Our library targets power-performance adaptation of OpenMP applications and is implemented using only portable components. To use our library, applications are instrumented with function calls around each adaptable phase, which are delimited as OpenMP parallel regions. At runtime, the library controls the execution of each phase in terms of the number of threads, their placement on cores, and the DVFS level selected by the predictor for global use. During the sampling phase, configurations are set appropriately and event rates are collected automatically by the library. We use the *omp_set_num_threads()* call to set concurrency within OpenMP

and thread bindings with the Linux processor affinity system call, *sched_setaffinity()*. We record hardware event rates with PAPI [2]. We set DVFS levels using the *cpufrequtils* library, specifically using the *sysfs_set_frequency()* call. After making predictions, the library uses the predicted optimal configurations for all subsequent traversals of each phase.

Several forms of adaptation are possible through our runtime library. The simplest ones optimize either DCT or DVFS but not both during a given run by using the corresponding model to predict the effects of the selected power-performance knob. We consider two mechanisms to adapt both DCT and DVFS in a single program run. First, we apply the two individual models sequentially to adapt first concurrency and then apply DVFS accordingly on the cores that are kept active. We refer to this model as the *sequential prediction model*. Second, we create a new model that simultaneously predicts changes in both concurrency and frequency, which we refer to as the *unified prediction model*.

When adapting DCT, the library can compare configurations simply using the predicted $uIPC$. However, when considering DVFS, including the hybrid approaches, we must be careful to ensure valid performance comparisons. A problem arises here because at lower frequencies each cycle lasts longer, which causes higher IPCs to occur at lower frequencies while the program actually runs slower. For this reason, we calculate instructions per second before making comparisons using the known frequency levels.

A program may have phases that are of too fine granularity to benefit from adaptation using either DVFS or DCT, as the overhead of performing adaptation can exceed its benefits. We have empirically identified a threshold of one million cycles, below which we simply use the currently active configuration when entering a phase. In practice, most application phases are much longer than the selected threshold; however short phases do exist and may distort performance significantly, if their locally optimal configurations differ from the optimal configurations of adjacent dominant phases.

5. EXPERIMENTAL ANALYSIS

In this section, we evaluate our multi-dimensional prediction model. We begin with a brief description of the experimental setup. Next, we analyze the scalability of the benchmarks on our target machine. Then, we evaluate the model of performance prediction used to apply DVFS and DCT. Finally, we compare the benefits of applying DVFS and DCT independently and synergistically, in terms of both performance and energy benefits.

5.1 Experimental Setup

Our experimental platform has two Intel Xeon E5320 quad-core processors, for a total of eight cores. Each of two pairs of cores within a chip shares a 4MB L2 cache, creating an asymmetry in scheduling decisions in that two threads can be scheduled on a single chip in two different ways, with cache sharing and without it. Each core operates at a maximum frequency of 1.86GHz, with the possibility of reducing to 1.60GHz. The system has a 1066 MHz FSB, contains 4GB of memory, and runs Linux kernel version 2.6.22. In all experiments, full system energy is collected per run using a Watts Up Pro power meter, and the average power consumption is computed based on the execution time and total energy consumption.

We experimented with benchmarks representative of parallel applications from the HPC domain. Specifically, we use seven benchmarks from the OpenMP version of the NAS Parallel Benchmarks suite (3.1), compiled at class size B. The benchmarks have large variation in several interesting execution properties, including number of phases, scalability (global and per phase), compute- and memory-boundedness of phases, number of loop iterations, and computational intensity, thus making prediction challenging.

5.2 Application Scalability Analysis

Before evaluating DVFS- and DCT-based adaptation using performance prediction, we briefly analyze the scalability of the applications on our platform. To do so, we execute each application under all symmetric configurations on the experimental platform and record the execution time. Figure 1 presents the scalability results on our dual-processor, quad-core Intel Xeon system. As stated earlier, two threads on a single chip can execute with shared or private caches on this architecture. We use the notation $2s$ to indicate a shared cache and $2p$ to indicate private caches on our graphs. The notation (X, Y) denotes non-adaptive execution with X processors and Y cores per processor, and later an additional term Z is included to indicate the DVFS level used.

Figure 1 shows that, in general, applications are far from scaling perfectly on the target platform. In particular, only one application achieves its best performance using all 8 cores. We observe essentially three categories of scalability in our experiments. First are those applications that manage reasonable speedup through the utilization of additional cores (BT, FT, and UA). Second are applications that incur a non-negligible performance *loss* when using *more* cores (IS and MG). Third are applications that neither substantially gain or lose performance from higher concurrency (CG and SP). Energy consumption generally increases with more cores.

The most energy-efficient configuration coincides with the most performance-efficient configuration for 4 out of the 7 benchmarks (BT, CG, FT, and IS). For 3 benchmarks (MG, UA, SP), the user can use fewer than the performance-optimal number of cores, to achieve substantial energy savings, at a marginal performance loss. We also observe that for a given number of threads, performance can be very sensitive to the mapping of threads to cores (e.g. BT, FT, and SP when executed with 2 or 4 threads). Even if performance is insensitive to the mapping of threads to cores, power can be sensitive to the mapping of threads to cores. In MG, for example, distributing two threads onto cores that do not share L2 cache, but are on the same die, is imperceptibly less performance-efficient, but significantly more energy-efficient than distributing two threads across two processors.

We attribute the observed poor scalability of several benchmarks to memory contention, at all levels of the memory hierarchy. Specifically, two cores sharing a cache rarely benefit from data sharing, but rather suffer from destructive interference in the form of increased conflict misses between threads. Further, additional threads produce a higher demand on main memory, producing contention at the shared front-side bus. These issues combine to limit scalability most in applications that are memory intensive and have primary or secondary working sets too large to fit into on-chip cache space.

5.3 Performance Prediction Evaluation

We selected a single benchmark to train the model, trading potentially higher prediction accuracy for less training time. Specifically, we used NAS-UA to perform training. UA has a large number of phases and widely varying execution characteristics on a per phase basis, including IPC, scalability, locality, and granularity. We have also used extended training sets with more benchmarks, but we do not present those results, since they did not notably improve model accuracy compared to our reduced training set. We selected sample configurations for each model to maximize the amount of information available to the model. For the DVFS model, we se-

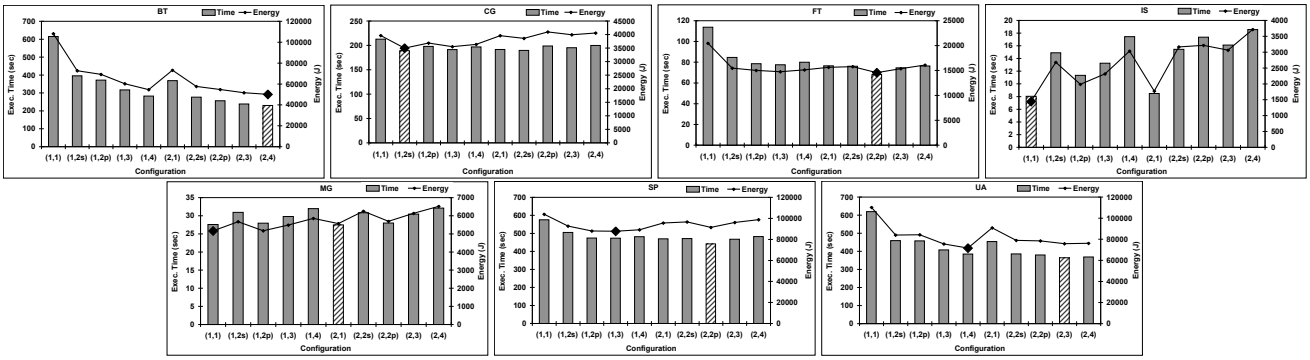

Figure 1: Execution time (bars) and energy consumption (lines) of the benchmarks across all configurations. The notation (X, Y) denotes non-adaptive execution with X processors and Y cores per processor. The configurations with the best performance and energy for each benchmark are marked with stripes and a diamond respectively.

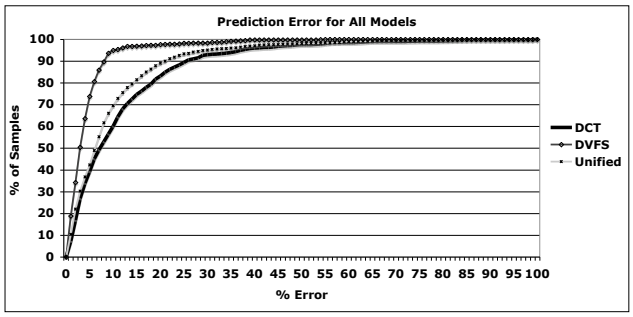

Figure 2: Cumulative distribution functions of prediction accuracy of the three prediction models.

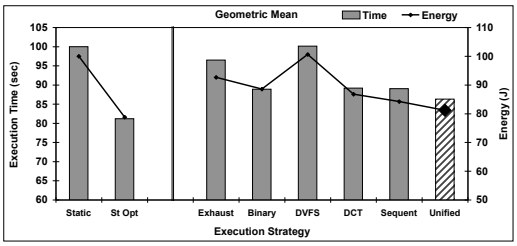

Figure 4: Geometric means of the benefits of adaptation through various strategies. The adaptive strategies with the best mean performance and energy are marked with stripes and a large diamond respectively.

lected a single sample at maximum frequency within each given threading configuration. We selected two samples for DCT: (1,3) and (2,4). Finally, for the unified DVFS-DCT model, we selected three sample configurations: (1,2p,2), (2,2s,1), and (2,4,2), where the third term indicates the frequency level with the lower number representing the lower frequency. These sample configurations were selected as outlined in Section 3.6 to provide data along each dimension of adaptation. In all cases, we made predictions for all configurations not sampled.

Using many events can benefit the model, however current processors severely limit the maximum number of events that we simultaneously record, while multiplexing many events on the available event registers has a significant overhead and limited accuracy. Thus, we set the number of events used in our model to the number supported in the hardware. On our experimental platform, only two event registers are available, and one must always be used to collect $uIPC$ which is mandatory with our model. For all three models, the statistically selected auxiliary event with the highest correlation with target IPC in the training data was L1 data cache accesses.

We derived the model coefficients offline using linear regression on samples of event rates and $uIPC$ on each configuration from the training benchmark. Figure 2 shows the percent of predicted samples for each model with error less than a particular threshold indicated on the x-axis. The results demonstrate high accuracy of the model in all three cases. In particular, the DVFS model yields a median error of only 3.0% (4.2% mean), the DCT model a median of 7.3% (11.2% mean), and the unified model a median of 6.1% (9.5% mean). We note that prediction is performed with input from 1, 2, or 3 sample configurations for the remaining of 20 possible configurations on our platform. The higher accuracy of predicting DVFS than DCT results from a simpler set of effects in changing DVFS levels that our model captures easily, while DCT has complicated performance effects, due to the irregular, non-monotonic scalability patterns of many phases. Of the 20 possible configurations, the unified model correctly identifies the single best configuration in 35% of phases, one of the top three in 51.3% of phases, and in only 7% of phases are any of the ten worst configurations incorrectly selected. The predicted optimal configuration is on average 6.1% slower than the true optimal configuration. These results indicate that the model is an accurate means of attaining performance estimates to tune power-performance parameters without requiring potentially expensive empirical searches. The results compare favorably with similar empirical models [6, 16].

5.4 Evaluation of DCT and DVFS Adaptation

In this section, we evaluate the use of our prediction models in conjunction with runtime adaptation on multithreaded scientific codes. We begin by comparing the use of only DVFS or DCT. We then analyze two schemes for adapting both DVFS and DCT, specifically applying them sequentially or in a unified manner. Finally, we compare prediction-based adaptation against the use of empirical search in identifying optimal configurations. Figure 3 presents the results of adaptation through the various mechanisms for each benchmark, Figure 4 shows the geometric mean of the normalized energy and execution time results, and Figure 5 gives the geometric mean of ED^2. ED^2 represents the product of energy consumption and the square of execution time, which is a common metric for energy-efficiency in high performance computing.

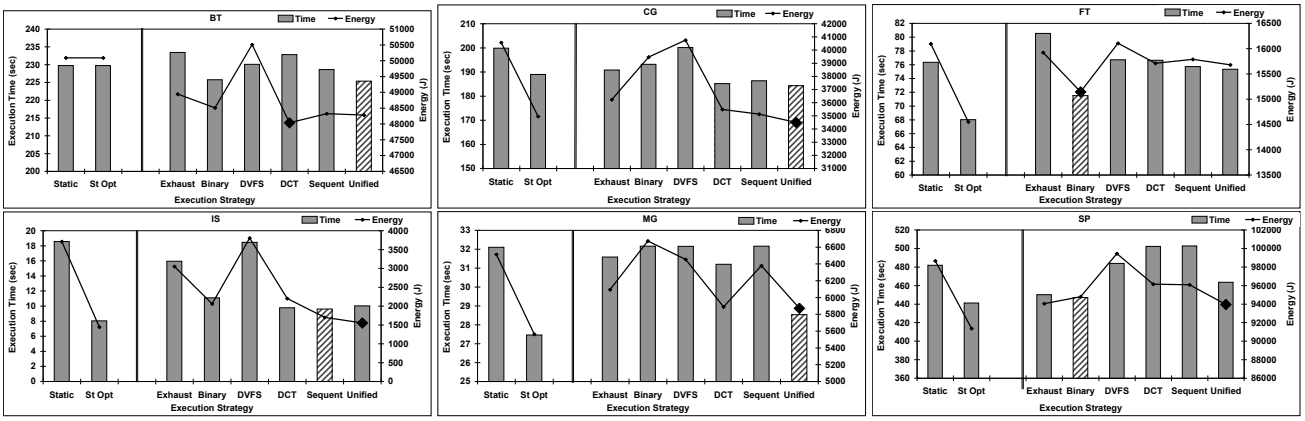

Figure 3: Results of adaptation through various techniques. The group of bars left of the divider represent static configurations and those right of the divider are the adaptive strategies. The adaptive strategies with the best mean performance and energy are marked with stripes and a large diamond respectively.

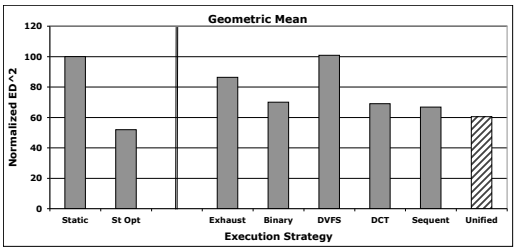

Figure 5: Geometric means of the benefits of adaptation through various strategies. The adaptive strategy with the best mean ED^2 is marked with stripes.

We compare the various strategies against the results of "static executions" that use a single configuration for the entire execution. We specifically compare to using full concurrency and frequency (*static*) and to the best performing of all static executions (*static optimal*). Clearly, we derive the static optimal post-facto: we could not know it online in practice without exhaustive offline execution of each application on all configurations, for each specific input. We use static optimal as a potentially unrealistic, baseline of comparison for the other strategies. The static optimal, however, is not necessarily the overall optimal execution point, as each phase may have its own dynamic optimal configuration. We do not consider this possibility as its identification requires exponential time, making it unrealistic even for offline use.

5.4.1 Adaptation using one knob

In this analysis, we make adaptation decisions by selecting the configuration with the highest predicted performance, because HPC applications in general must maintain maximum performance. The results of applying DVFS using the model support the intuition that DVFS is generally unable to improve performance compared to simply using all available cores at maximum frequency. The literature includes corner cases of memory-bound phases where this assumption is violated and scaling down frequency can marginally improve execution time [8], but these phases are rare exceptions. Specifically, our experiments reveal no benefit in terms of performance or energy from adapting to the DVFS level with the highest predicted performance. Without allowing for some loss in performance, DVFS does not generally benefit energy consumption.

On the other hand, using DCT with the prediction model provides substantial benefits in execution time (10.8% mean savings), power consumption (2.7% mean savings), energy consumption (13.2% mean savings), and ED^2 (31.0% mean savings) compared to the static execution with all cores active. Despite the advantages of DCT, mispredictions for two benchmarks result in an observed increase in execution time—BT by 1.3% and SP by 4.2%. However, both benchmarks still manage energy savings of 4.1% and 2.6%, respectively, because of the reduced power consumed by the fewer active cores. In contrast, the largest benefit occurs with IS which sees a 40.7% reduction in energy consumption. When compared to the static optimal execution, DCT is within 8.0% mean performance and even surpasses that performance with CG by 2.0% due to phase-awareness. These results indicate that at least at the scale of a few multi-core processors integrated on a single node, DCT is the more suitable of the two power-performance knobs for use in the HPC domain, since it can improve performance and energy-efficiency by substantial margins simultaneously.

5.4.2 Adaptation using two knobs

We can combine the two power-performance knobs in multiple ways. First, we consider applying them sequentially. We first apply DCT and then DVFS on the active cores in each phase rather than the other way around, since DCT has a clear advantage over DVFS in reducing power while improving performance, as exhibited in our earlier experiments. We note that this may not be necessarily true in other experimental platforms that allow more fine-grain control of voltage/frequency. Since we make decisions to maximize predicted IPC and our platform has only two DVFS levels with a small frequency difference, DVFS adds very little benefit to DCT alone, and no reduction in execution time compared to DCT alone occurs. However, DVFS reduces power and energy consumption by 2.7% and 2.6% respectively beyond DCT alone on average, by identifying several isolated phases to reduce frequency without negatively impacting performance, resulting in a cumulative mean improvement in ED^2 of 34.3% relative to using all cores.

The major advantage of the unified prediction approach is that it eliminates blind-spots in the configuration space during the prediction process. Whereas sequential application of DVFS and DCT will only evaluate DVFS options on the decided DCT level, the unified approach considers all possible values of each parameter at a single stage. Further, the unified scheme uses the same number of

execution samples as the sequential approach, however it uses all samples for both DVFS and DCT models, instead of dividing them.

Because of its advantages over the sequential approach, the unified scheme improves performance by 2.8% and reduces energy by 3.0% (geometric mean improvements over the sequential approach). When compared to the default execution using maximum concurrency and frequency, the advantages of unified adaptation become even clearer. Specifically, we see an 13.7% speedup simultaneous with a 5.9% reduction in power consumption, resulting in an overall reduction in energy consumption of 18.8% and ED^2 of 39.5% (geometric mean improvements over static execution on all cores). In fact, all benchmarks experience improved performance and reduced energy consumption, and the unified scheme achieves the lowest execution time in three of six cases and the lowest energy consumption in four of six cases. Even when compared to the oracle-derived executions on the static optimal configuration for each benchmark, unified adaptation achieves energy consumption within 2.4% and performance within 5.1% (geometric means), and better performance by 2.3% in the case of BT due to identification of improved per-phase configurations. This indicates that prediction models are both viable and effective in addressing the multi-dimensional adaptation problem.

5.4.3 Comparison with existing DVFS approach

Ondemand is an existing tool for automatically applying DVFS that also uses hardware performance monitors for guidance [23]. *Ondemand* is an in-kernel tool that works by monitoring dynamic application CPU utilization to reduce processor frequency at times of low load and uses the same *frequtils* interface as we exploit. *Ondemand*, and most other similar tools, uses Intel ACPI, which is an interface to allow changes in DVFS levels and processor power states when idle, with transition criteria determined by the adaptation system. We applied *Ondemand* in modes similar to our DVFS adaptation only and our Sequential strategies, through which we achieved similar results (within 1% for energy consumption and nearly identical run time). These results demonstrate that our DVFS scheme is competitive with state of the art DVFS tools. More importantly, it does not alter our fundamental observation that integrated DCT and DVFS adaptation provides the best overall results.

5.4.4 Prediction vs. search approaches

We have also implemented two empirical search approaches to identify optimal DVFS and DCT configurations. The first of these performs an exhaustive search of the configuration space before making a decision, while measuring the execution time of phases with each configuration. This approach does not require any offline training, so the programmer can use it with minimal effort. However, the online overhead of testing many possible configurations stands to reduce any potential benefit of adaptation considerably, which is what occurs in practice. The exhaustive search method reduces execution time by 3.5%, power by 4.0%, energy by 7.3%, and ED^2 by 13.7%, well below the savings of our prediction-based techniques. Exhaustive search proves superior to prediction schemes in SP, which executes 400 workload-invariant iterations.

For comparison purposes we also consider a heuristic search approach, based on a binary search of the configuration space, similar to the approach evaluated by Li and Martinez [18]. A fair direct comparison between our prediction models and the approach discussed previously [18] is not possible, since contrary to the simulation-based study in previous work, our evaluation was conducted on a real system with a different workload (NAS benchmarks), and adaptation through binary search was implemented by timing the execution time of phases during a single run instead of over multiple runs of each benchmark. Nevertheless, we believe that our comparison can still provide some useful insight on the appropriateness of heuristic search and prediction-based approaches to dynamic program adaptation.

Our implementation of binary search begins by executing at full concurrency and frequency, then sequentially performs binary searches of the concurrency and DVFS dimensions. During the searches, if a sample is tested with worse performance than the first sample, concurrency or DVFS is increased in the next tested sample. This approach has considerably reduced overhead compared to the exhaustive search, because many configurations need not be tested, resulting in 7.6% better performance and 4.1% lower energy consumption (geometric mean improvements over exhaustive search). Compared to the static execution, performance is improved by 11.1%, energy by 11.4%, and ED^2 by 30.0%, however power consumption is only reduced by 0.4%. This suggests that a heuristic search can be effective in the context of adapting DCT and DVFS at runtime. However, it still falls short of the static optimal configuration by 7.7% for performance and 9.8% for energy.

The most interesting comparison is between the unified prediction model and binary search. Binary search achieves performance 2.6% worse than the unified prediction approach while consuming 7.4% more energy and seeing a 9.6% increase in ED^2 (geometric mean differences). Binary search suffers from blind-spots that prevent identification of effective configurations at low concurrency or DVFS levels, which tend to consume less power, so the unified prediction model can reduce power further, on average by 5.5%. Binary search does have better performance than the unified model in two of six cases (FT and SP), however energy consumption is higher in all but one case (FT). In particular, the results of binary search suffer for MG and IS because they contain too few iterations to amortize the search overhead, in contrast to BT and SP, where binary search excels since the applications execute 200 and 400 iterations respectively. As future systems continue to increase in parallelism as well as the number of DVFS levels available, the overhead of searching is expected to increase and the relative benefit of prediction is expected to grow.

6. RELATED WORK

Research on software-controlled dynamic power management has focused extensively on controlling voltage supply and frequency in single-core processors. This research has derived analytical models for DVFS [28], compiler-driven techniques [29], and control-theoretic approaches [26]. Similar techniques have been employed to reduce dynamic power management in system components other than processors, such as RAM [7] and disk [3]. Researchers have recently modeled and analyzed the impact of a single control knob, either DVFS or concurrency throttling, on dynamic power management on shared-memory [6, 11, 20, 24], and on distributed-memory parallel systems [8, 9, 25].

Our work differs from earlier research on power-aware adaptation using a single knob in several key aspects. First, it achieves two-dimensional adaptation. Second, it leverages a scalable performance prediction model, instead of direct measurements or static analysis of idle execution intervals. Third, it analyzes the busy intervals of parallel computation to exploit opportunities for power savings and performance improvement simultaneously, as opposed to exploiting only slack time to reduce power. Fourth, it uses a model that is general and versatile: it can accommodate different optimization targets—both performance-centric and energy-centric—with ease and it is developed with an automated and portable methodology. In terms of actual implementation, the proposed model leverages phase-aware adaptation at the granularity of

parallel loops, which has been explored before in compiler-based DVFS algorithms for multiprocessors [20, 29].

Prediction models for adaptation via concurrency throttling were introduced by Curtis-Maury, et al [5, 6]. The current work makes several new contributions in the context of performance prediction for power-performance adaptation. We consider prediction models for DVFS and DCT simultaneously, effectively exploring a larger and more challenging runtime optimization space. We draw comparisons between alternative power-performance adaptation methods and present effective strategies to synthesize multiple power-performance adaptation methods in software. Furthermore, we propose methods to generalize multi-dimensional prediction models using sampling of the target configuration space and significantly improve prediction accuracy compared to previous work [5, 6], thus achieving better optimization with zero tolerance for performance loss. We improve earlier regression models for cross-configuration prediction [5, 6] through such techniques as variance stabilization, explicit consideration of event and configuration interactions, and architecture-aware sampling.

Our contribution shares similar objectives with research presented by Li and Martinez [18], whose work evaluated search algorithms for DVFS and DCT, so as to meet specific performance targets under a given power budget. Our work differs in that it uses statistical prediction models instead of direct search methods and that it considers only power-performance adaptation schemes that do not penalize performance, effectively targeting the more performance-sensitive HPC environments. The use of prediction makes our contribution an attractive alternative for runtime adaptation, since the number of hardware event counter samples needed to predict across all concurrency and voltage/frequency configurations of a system can be very small (2–3) compared to the samples needed by any search strategy. Thus, the performance of prediction-based adaptation scales more gracefully with the number of cores and voltage-frequency levels than search methods, while being highly competitive at small system scales.

Several researchers have previously explored the use of hardware event counters for characterizing performance and power properties [12, 21, 27]. Our models differ in that they provide cross-configuration predictions with multi-dimensional inputs, using hardware event counters. As such, they achieve accurate statistical correlation between event samples and performance, across a potentially large and hard to search system configuration space. In this respect, our work is more closely related to regression and machine learning methods for performance prediction and design space exploration on parallel architectures [16, 17]. Both our contribution and earlier regression-based performance prediction methods use statistical analysis of the correlation between multiple parameters and performance. The key difference is that our framework is used for online workload adaptation, rather than for off-line exhaustive exploration. Our prediction model is simpler than models used in design space exploration, however it is fast enough to use in runtime optimization.

7. CONCLUSIONS

The number of cores integrated in a single processor is increasing at an exponential rate; however most applications, even from the highly specialized HPC domain, can hardly exploit many cores. We have shown that HPC applications observe performance losses even beyond modest concurrency levels on an 8-core system. We presented a model to predict the performance effects of applying multiple energy saving techniques simultaneously. The model applies statistical analysis of hardware event rates to estimate how voltage/frequency scaling and dynamic concurrency throttling influence performance in application phases and across system configurations. Over a range of benchmarks, our model achieves a median error of only 6.1% in prediction, in response to simultaneous tuning of DVFS and DCT. The high prediction accuracy allows for the successful identification of efficient operating points and phase-aware adaptation in HPC applications.

We have applied our model to adapt program execution by regulating concurrency in conjunction with thread placement, as well as DVFS levels. Our results indicate that while DVFS on its own is not ideal for the HPC domain where performance is critical, DCT is an attractive solution. Further, we find that combining the two approaches in a synergetic fashion, can simultaneously improve performance and energy-efficiency relative to either approach in isolation. Specifically, a unified adaptation model achieves performance improvements of 14%, power savings of 6%, energy savings of 19%, and a 40% reduction in ED^2 compared to using all cores at full frequency, outperforming an approach which sequentially applies DCT and DVFS. We also compare our prediction model to methods using exhaustive or binary search that time system configurations. We find that while binary search outperforms exhaustive search, it is inferior to the prediction-based approach due to overhead and blind-spots. As we scale to more cores and DVFS levels, the overhead of search-based approaches is likely to increase, widening the advantage of prediction. Since the performance of prediction-based methods can effectively approximate that of an oracle, we conclude they are a viable alternative for future-generation systems with many cores and fine-grain power control capabilities.

Our work is not without limitations, which we plan to address in future research. A linear regression model achieves low overhead for runtime adaptation, at the cost of accuracy. More elaborate models, such as piecewise polynomial approximation or neural networks, may improve prediction accuracy, at the cost of increased runtime overhead. A detailed analysis of this trade-off is needed to draw more accurate conclusions.

Adaptation schemes have both direct and indirect costs while switching system configurations. Direct costs stem from the actual switching overhead, while indirect costs stem from gradual redistribution of the working set of the application between cores and caches. Our prediction model currently does not account explicitly for any indirect costs of adaptation. Both the selection of samples during training/actuation and the configuration interaction terms in the model need to be revisited to incorporate interference due to changing configurations between adjacent phases. Preliminary investigation shows that although cross-phase interference is not acute on small-scale multi-core systems, it is far more noticeable on large-scale scalable systems, such as NUMA platforms.

Adaptation capabilities are not readily available in all applications, as they are often prohibited by the semantics of the programming environment. For example, MPI applications are typically much harder to implement adaptively than OpenMP applications. Addressing this issue will require efforts to make parallel programming runtime environments more amenable to dynamic concurrency throttling. A readily available but inefficient DCT solution for MPI applications is the use of core overloading (i.e. mapping more than one processes per core).

Acknowledgments

This research is supported by grants from NSF (CCR-0346867, CCF-0715051, CNS-0521381, CNS-0720750, CNS-0720673), the U.S. Department of Energy (DE-FG02-06ER25751, DE-FG02-05ER25689), IBM, and Virginia Tech (VTF-874197). Partly performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. (UCRL-CONF-400453).

8. REFERENCES

[1] M. Azimi, N. Cherukuri, D. Jayashima, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. Vaidya. Integration Challenges and Tradeoffs for Tera-scale Architectures. *Intel Technology Journal*, August 2007.

[2] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In *Proc. of Supercomputing'2000*, November 2000.

[3] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *Proc. of the 17th International Conference on Supercomputing*, June 2003.

[4] K. Chakraborty, P. Wells, and G. Sohi. A Case for an Over-provisioned Multicore System: Energy Efficient Processing of Multithreaded Programs. Technical Report TR-1607, Department of Computer Sciences, University of Wisconsin-Madison, 2007.

[5] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos. Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes. *IEEE Transactions on Parallel and Distributed Systems*. Accepted, to appear, 2008.

[6] M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos. Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction. In *Proc. of the International Conference on Supercomputing*, June 2006.

[7] B. Diniz, D. O. G. Neto, W. Meira Jr., and R. Bianchini. Limiting the Power Consumption of Main Memory. In *Proc. of the International Symposium on Computer Architectures*, June 2007.

[8] R. Ge, X. Feng, and K. W. Cameron. Performance constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters. In *Proc. of Supercomputing*, November 2005.

[9] C.-H. Hsu and W. Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Proc. of Supercomputing'05*, November 2005.

[10] C. Huang, O. Lawlor, and L. Kale. Adaptive MPI. In *Proc. of the 16th International Workshop on Languages and Compilers for Parallel Computing, LNCS 2948*, 2003.

[11] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proc. of the International Symposium on Microarchitecture*, December 2006.

[12] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proc. of the Annual International Symposium on Microarchitecture*, December 2003.

[13] E. Joseph, A. Snell, C. G. Willard, S. Tichenor, D. Shaffer, and S. Conway. Council on Competitiveness Study of ISVs Serving the High Performance Computing Market. July 2005.

[14] T.S. Karkhanis and J.E. Smith. A First-Order Superscalar Processor Model. In *Proc. of the 31st International Symposium on Computer Architecture*, June 2004.

[15] S. Kumar, H. Raj, K. Schwan, and I. Ganev. Re-architecting VMMs for Multicore Systems: The Sidecore Approach. In *Proc. of the 2007 Workshop on the Interaction between Operating Systems and Computer Architecture*, June 2007.

[16] B. C. Lee and D. M. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2006.

[17] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee. Methods of Inference and Learning for Performance Modeling of Parallel Applications. In *Proc. of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, November 2007.

[18] J. Li and J. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multi-Processors. In *Proc. of the International Symposium on High Performance Computer Architecture*, February 2006.

[19] Y. Li and B. C. Lee and D. Brooks and Z. Hu and K. Skadron. CMP Design Space Exploration Subject to Physical Constraints. In *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, February 2006.

[20] C. Liu, A. Sivasubramaniam, M. T. Kandemir, and M. J. Irwin. Exploiting Barriers to Optimize Power Consumption of CMPs. In *Proc. of the 19th International Parallel and Distributed Processing Symposium*, April 2005.

[21] A. Merkel and F. Bellosa. Balancing Power Consumption in Multiprocessor Systems. In *Proc. of EuroSys Conference*, April 2006.

[22] T. Moseley, J. Kim, D. Connors, and D. Grunwald. Methods for Modeling Resource Contention on Simultaneous Multithreaded Processors. In *Proc. of the 2005 International Conference on Computer Design*, October 2005.

[23] V. Pallipadi and A. Starikovskiy. The Ondemand Governor. In *Proc. of the Ottawa Linux Symposium*, July 2006.

[24] S. Park, W. Jiang, Y. Zhou, and S. V. Adve. Managing Energy-Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures. In *Proceedings of the 2007 ACM SIGMETRICS*, June 2007.

[25] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster. In *Proc. of the Symposium on Principles and Practice of Parallel Programming*, March 2006.

[26] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, and B. L. Jacob. A Control-Theoretic Approach to Dynamic Voltage Scheduling. In *Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, October 2003.

[27] A. Weissel and F. Bellosa. Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management. In *Proc. of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems*, October 2002.

[28] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004.

[29] Q. Wu, M. Martonosi, D. Clark, V. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. Dynamic Compiler-Driven Control for Microprocessor Energy and Performance. *IEEE Micro*, 26(1), 2006.