Hybrid MPI/OpenMP Power-Aware Computing

Dong Li[†] Bronis R. de Supinski[⋆] Martin Schulz[⋆] Kirk Cameron[†] Dimitrios S. Nikolopoulos[‡]

[†] Virginia Tech
Blacksburg, VA, USA
{lid,cameron}@cs.vt.edu

[⋆] Lawrence Livermore National Lab
Livermore, CA, USA
{bronis,schulzm}@llnl.gov

[‡] FORTH-ICS and University of Crete
Heraklion, Crete, GREECE
dsn@ics.forth.gr

***Abstract*—Power-aware execution of parallel programs is now a primary concern in large-scale HPC environments. Prior research in this area has explored models and algorithms based on dynamic voltage and frequency scaling (DVFS) and dynamic concurrency throttling (DCT) to achieve power-aware execution of programs written in a single programming model, typically MPI or OpenMP. However, hybrid programming models combining MPI and OpenMP are growing in popularity as emerging large-scale systems have many nodes with several processors per node and multiple cores per processor. In this paper we present and evaluate solutions for power-efficient execution of programs written in this hybrid model targeting large-scale distributed systems with multicore nodes. We use a new power-aware performance prediction model of hybrid MPI/OpenMP applications to derive a novel algorithm for power-efficient execution of realistic applications from the ASC Sequoia and NPB MZ benchmarks. Our new algorithm yields substantial energy savings (4.18% on average and up to 13.8%) with either negligible performance loss or performance gain (up to 7.2%).**

***Keywords*-MPI; OpenMP; performance modeling; power-aware high-performance computing.**

I. Introduction

The large energy footprint of high-end computing systems motivates holistic approaches to energy management that combine hardware and software solutions. Thus, software-controlled power-aware execution of HPC applications on large-scale clusters has become an important research topic [1], [2], [3], [4]. Most researchers have focused on processor-level power management schemes since processors dominate power consumption in HPC environments.

Two primary power-aware computing approaches currently exist for large-scale systems. Many state-of-the-art algorithms for software-controlled dynamic power management [5], [6], [7], [8] use dynamic voltage and frequency scaling (DVFS) to dilate computation into slack (any non-overlapped hardware or algorithmic latency) that occurs between MPI communication events, thus reducing energy consumption. Alternatively, dynamic concurrency throttling (DCT) [9], [10] controls the number of active threads executing pieces of parallel code, particularly in shared-memory programming models like OpenMP, to save energy and to improve performance simultaneously [11].

These software-controlled power-aware execution schemes for HPC applications have been integrated within a single parallel programming model, such as MPI or OpenMP. However, none have been applied to applications written in hybrid programming models, such as MPI/OpenMP. Since multicore nodes with larger core counts and less memory per node are becoming prevalent, we anticipate hybrid programming models will become common. Hybrid programming models complicate power management since any solution must consider inter-node and intra-node effects simultaneously. Thus, several new research issues arise when applying DCT and DVFS to hybrid programming models. We explore these issues in the context of the hybrid MPI/OpenMP programming model, to provide answers to the following questions:

- Does using DCT within one MPI task affect the execution in other tasks?
- How can we identify slack due to intra- and inter-node interactions in hybrid programs?
- How should we coordinate DCT and DVFS to save energy?

We contribute a new power-aware modeling methodology for the hybrid MPI/OpenMP model. Based on it, we design and implement a power-aware runtime library that adaptively applies DVFS and DCT to hybrid MPI/OpenMP applications. Our modeling approach and runtime library implementation answer the above research issues. Our main contributions are:

- A formalization of the interactions between MPI tasks under DCT control that identifies the impact of DCT on other tasks;
- A novel DCT coordination scheme;
- A new analysis of the implicit penalty of concurrency throttling on last-level cache misses and a DCT algorithm that aggregates OpenMP phases to overcome this problem;
- A unified intra- and inter-node method to identify the slack available for DVFS control in hybrid applications;
- A novel combined DCT/DVFS system in which the DVFS scheduler accounts for the effects of DCT including explicit prediction of the time required for OpenMP phases at different concurrency levels;
- A study of power-saving opportunities in both strong and weak scaling of hybrid applications at unprecedented system scales of up to 1024 cores.

Our results, obtained with applications from the ASC Se-

978-1-4244-6443-2/10/$26.00 ©2010 IEEE

quoia and the NAS Parallel Benchmark Multizone suites, on two systems with relatively wide shared-memory nodes (8 and 16 cores per node) show that our power-aware runtime library leverages the energy-saving opportunities in hybrid MPI/OpenMP applications while maintaining performance. Our scaling study demonstrates that power saving opportunities continue or increase under weak scaling but diminish under strong scaling. Overall, our power-aware runtime library for hybrid programming models saves significant energy—4.2% on average and as much as 13.8% in certain cases—with either negligible performance loss or performance gain up to 7.2%.

The rest of this paper is organized as follows. Section II provides background terminology for this work. Section III presents our power-aware performance prediction model for hybrid MPI/OpenMP applications. Section IV presents our execution time prediction methodology for OpenMP phases under DCT and DVFS control. Section V presents our dynamic concurrency throttling schemes and Section VI presents our dynamic voltage and frequency scaling schemes for hybrid MPI/OpenMP applications. Section VII presents our experimental analysis. Section VIII discusses related work and Section IX concludes the paper.

II. Hybrid MPI/OpenMP Terminology

Large-scale system trends motivate our consideration of hybrid programming models. HPC systems are rapidly increasing in scale in terms of numbers of nodes, numbers of processors and numbers of cores per processor, with declining main memory and secondary cache sizes per core. These trends encourage the use of shared-memory models within a node to exploit fine-grain parallelism, to achieve better load balance, to reduce application memory footprints and to improve memory bandwidth utilization [12]. However, message-passing remains preferable between nodes since it simplifies minimization of communication overhead.

Most hybrid programming models exploit coarse-grain parallelism at the task level and medium-grain parallelism at the loop level. Thus, we consider programs that use the common THREAD_MASTERONLY model [13]. Its hierarchical decomposition closely matches most large-scale HPC systems, which are comprised of clustered nodes, each of which has multiple cores per node, distributed across multiple processors. In this model, a single master thread invokes all MPI communication outside of parallel regions. Almost all MPI programming environments support the THREAD_MASTERONLY model. OpenMP directives parallelize the sequential code of the MPI tasks. This solution exploits fast intra-task data communication through shared memory via loop-level parallelism. While other mechanisms (e.g., POSIX threads) could add multi-threading to MPI tasks, OpenMP supports incremental parallelization and, thus, is widely adopted by hybrid applications.

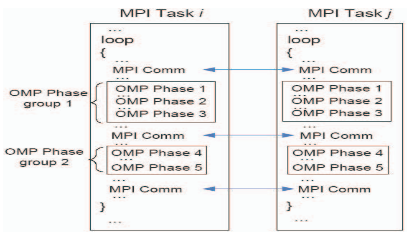


Figure 1: Simplified typical MPI/OpenMP scheme

Iterative parallel computations dominate the execution time of scientific applications. Hybrid programming models exploit these iterations. Figure 1 depicts a typical iterative hybrid MPI/OpenMP computation, which partitions the computational space into subdomains, with each subdomain handled by one MPI task. The communication phase (MPI operations) exchanges subdomain boundary data or computation results between tasks. Computation phases that are parallelized with OpenMP constructs follow the communication phase. We use the term *OpenMP phases* for the computation phases delineated by OpenMP parallelization constructs.

Collections of OpenMP phases delineated by MPI operations form *OpenMP phase groups*, as shown in Figure 1. Typically, MPI collective operations (e.g., MPI_Allreduce and MPI_Barrier) or grouped point-to-point completions (e.g., MPI_Waitall) delineate OpenMP phase groups. No MPI primitives occur within an OpenMP phase group. MPI operations may include slack since the wait times of different tasks can vary due to load imbalance. Based on notions derived from critical path analysis, the *critical task* is the task upon which all other tasks wait.

Our goal is to adjust *configurations* of OpenMP phases of hybrid MPI/OpenMP applications dynamically. A *configuration* includes CPU frequency settings and concurrency configurations. The *concurrency configuration* specifies how many OpenMP threads to use for a given OpenMP phase and how to map these threads to processors and cores. This can be done by OpenMP mechanisms for controlling the number of threads and by setting the CPU affinity of threads using system calls. We use DCT and DVFS to adjust configurations so as to avoid performance loss while saving as much energy as possible. Also, configuration selection should have negligible overhead. For this selection process, we sample selected hardware events during several iterations in the computation loop for each OpenMP phase, and collect timing information for MPI operations. From this data, we build a power-aware performance prediction model that determines configurations that—according to predictions—can improve application-wide energy-efficiency.

III. Power-aware MPI/OpenMP Model

Our power-aware performance prediction model estimates the energy savings that DCT and DVFS can provide for hybrid MPI/OpenMP applications. Table I summarizes the

M	Number of OpenMP phases in a OpenMP phase group
ΔE_{ij}^{dct}	Energy saving by DCT during OpenMP phase j of task i
x_{ij},y_{ij}	Number of processors (x_{ij}) and number of cores per processor (y_{ij}) used by OpenMP phase j of task i
X,Y	Maximum available number of processors (X) and number of cores (Y) per processor on a node
T_{ij}	Time spent in OpenMP phase j of task i under a configuration using X processors and Y cores per processor
t_{ij}	Time spent in OpenMP phase j of task i after DCT
t_i	Total OpenMP phases time in task i after DCT
$t_{i,j,thr}$	Time spent in OpenMP phase j of task i using a configuration thr with thread count $\|thr\|$
N	Number of MPI tasks
f_0	Default frequency setting (highest CPU frequency)
Δt_{ijk}	Time change after we set frequency f_k during phase j of task i

Table I: Power-aware MPI/OpenMP model notation

notation of our model. We apply the model at the granularity of OpenMP phase groups. OpenMP phase groups exhibit different energy-saving potential since each group typically encapsulates a different major computational kernel with a specific pattern of parallelism and data accesses. Thus, OpenMP phase groups are an appropriate granularity at which to adjust configurations to improve energy-efficiency. We discuss the implications of adjusting configurations at the finer granularity of OpenMP phases in Section V.

DCT attempts to discover a concurrency configuration for an OpenMP phase group that minimizes overall energy consumption without losing performance. Thus, we prefer configurations that deactivate complete processors in order to maximize the potential energy savings. The energy saving achieved by DCT for task i is:

$$\Delta E_i^{dct} = \sum_{1 \le j \le M} \Delta E_{ij}^{dct}, \qquad (1)$$

Where ΔE_{ij}^{dct} is the energy savings for phase j relative to using all cores, when we use $x_{ij} \le X$ processors and $y_{ij} \le Y$ cores per processor. If the time for phase j, t_{ij}, is no longer than the time using all cores T_{ij}, as we try to enforce, then $\Delta E_{ij}^{dct} \ge 0$ and DCT saves energy without losing performance.

Ideally, DCT selects a configuration for each OpenMP phase that minimizes execution time and, thus, the total computation time in any MPI task. We model this total execution time of OpenMP phases in MPI task i as:

$$t_i = \sum_{j=1}^{M} \min_{1 \le |thr| \le X \cdot Y} t_{i,j,thr} \qquad (2)$$

The subscript thr in Equation (2) represents a configuration with thread count $|thr|$.

The critical task has the longest execution time, the *critical time*, which we model as:

$$t_c = \max_{1 \le i \le N} \sum_{j=1}^{M} \min_{1 \le |thr| \le X \cdot Y} t_{i,j,thr} \qquad (3)$$

The time difference between the critical task and other (non-critical) tasks in an OpenMP phase group is slack that we can exploit to save energy with DVFS. Specifically, we can use a lower CPU frequency during the OpenMP phases of non-critical tasks. These frequency adjustments do not incur any performance loss if the non-critical task, executed at the adjusted frequencies, does not spend more time inside the OpenMP phase group than the critical time. The *slack*

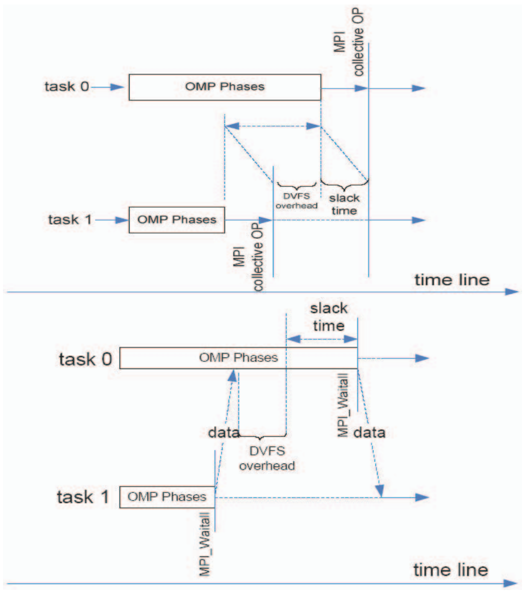


Figure 2: Leveraging slack to save energy with DVFS

time that we can disperse to the OpenMP phase group of a non-critical task i by DVFS is:

$$\Delta t_i^{slack} = t_c - t_i - t_i^{comm\_send} - t_{dvfs} \qquad (4)$$

Equation (4) reduces the available slack by the DVFS overhead (t_{dvfs}) and the communication time ($t_i^{comm\_send}$) for sending data from task i in order to avoid reducing the frequency too much. We depict two slack scenarios for MPI collective operations and MPI_Waitall in Figure 2. In each scenario, Task 0 is the critical task and Task 1 disperses its slack to its OpenMP phases.

We select a CPU frequency setting for each OpenMP phase based on the non-critical task's slack (Δt_i^{slack}). We discuss how we select the frequency in Section VI. We ensure that the selected frequency satisfies the following two conditions:

$$\sum_{1 \le j \le M} \Delta t_{ijk} \le \Delta t_i^{slack} \qquad (5)$$

$$\sum_{1 \le j \le M} t_{ijk} f_k \le t_i f_0 \qquad (6)$$

Equation (5) sets a time constraint: Δt_{ijk} refers to the time change after we set the frequency of the core or processor executing task i in phase j to f_k. Equation (5) requires that the total time changes of all OpenMP phases at the selected frequencies do not exceed the available slack we want to disperse. Equation (6) sets an energy constraint: t_{ijk} refers to the time taken by phase j of task i running at frequency f_k. We approximate the energy consumption of the phase as the product of time and frequency. Equation (6) requires that the energy consumption with the selected frequencies does not exceed the energy consumption at the highest frequency.

Intuitively, energy consumption is related to both time and CPU frequency. Longer time and higher frequency lead to more energy consumption. By computing the product of time and frequency, we capture the effect of both. Our energy estimation is not contradictory to previous CMOS models [5], [14], in which power and CPU frequency are related quadratically since we are estimating total system energy. Empirical observations [15] found average *system power* is approximately linear in frequency under a certain CPU utilization range. These observations support our estimate since HPC applications usually have very high CPU utilization under different CPU frequencies (e.g., all of our tests have utilization beyond 82.4%, well within the range of a near-linear relationship between frequency and system power).

IV. Time Prediction for OpenMP Phases

Our DVFS and DCT control algorithms rely on accurate execution time prediction of OpenMP phases in response to changing either the concurrency configuration or voltage and frequency. Changes in concurrency configuration should satisfy Equation 2. Changes in voltage and frequency should satisfy Equations 5 and 6.

We design a time predictor that extends previous work that only predicted IPC since intra-node DCT only requires a rank ordering of configurations [9], [11], [16]. We require time predictions in order to estimate the slack to disperse. We also require time predictions to estimate energy consumption. We use execution samples collected at runtime on specific configurations to predict the time on other, untested configurations. From these samples, our predictor learns about each OpenMP phase's execution properties that impact the time under alternative configurations. The input from the sample configurations consists of elapsed CPU clock cycles and a set of n hardware event rates ($e_{(1\cdots n,s)}$) observed for the particular phase on the sample configuration s, where the event rate $e_{(i,s)}$ is the number of occurrences of event i divided by the number of elapsed cycles during the execution of configuration s. The event rates capture the utilization of particular hardware resources that represent scalability bottlenecks, thus providing insight into the likely impact of hardware utilization and contention on scalability. The model predicts time on a given target configuration t, which we call $Time_t$. This time includes the time spent within OpenMP phases plus the parallelization overhead of those phases.

For an arbitrary collection of samples, S, of size $|S|$, we model $Time_t$ as a linear function:

$$Time_t = \sum_{i=1}^{|S|}(Time_i \cdot \alpha_{(t,i)}(e_{(1\cdots n,i)})) + \lambda_t(e_{(1\cdots n,S)}) + \sigma_t \quad (7)$$

The term λ_t is defined as:

$$\lambda_t(e_{(1\cdots n,S)}) = \sum_{i=1}^{n}(\sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,i,j,k)} \cdot e_{(i,j)} \cdot e_{(i,k)}))) + \sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,j,k,time)} \cdot Time_j \cdot Time_k)) + l_t \quad (8)$$

Equation (7) illustrates the dependency of terms $\alpha_{(t,i)}$, λ_t and σ_t on the target configuration. 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 and different frequency levels. The term $\alpha_{(t,i)}$ scales the observed $Time_i$ on the sample configurations up or down based on the observed values of the event rates in that configuration. The constant term σ_t is an event rate-independent term. It includes the overhead time for parallelization or synchronization. The term λ_t combines the products of each event across configurations and of $Time_{j/k}$ to model interaction effects. Finally, μ is the target configuration-specific coefficient for each event pair and l is the event rate-independent term in the model.

We use multivariate linear regression (MLR) to obtain the model coefficients (α, μ and constant terms) from a set of training benchmarks. We select the training benchmarks empirically to vary properties such as scalability and memory boundedness. The observed time $Time_i$, the product of the observed time $Time_i$ and each event rate and the interaction terms on the sample configurations are independent variables for the regression while $Time_t$ on each target configuration is the dependent variable. We derive sets of coefficients and model each target configuration separately.

We use the event rates for model training and time prediction that best correlate with execution time. We use three sample configurations: one uses the maximum concurrency and frequency, while the other two use configurations with half the concurrency—with different mappings of threads to cores—and the second highest frequency. Thus, we gain insight into utilization of shared caches and memory bandwidth while limiting the number of samples.

We verify the accuracy of our models on systems with three different node architectures. One has four AMD Opteron 8350 quad-core processors. The second has two AMD Opteron 265 dual-core processors. The third has two Intel Xeon E5462 quad-core processors. We present experiments with seven OpenMP benchmarks from the NAS Parallel Benchmarks suite (v3.1) with CLASS B input. We collect event rates from three sample configurations and make time predictions for OpenMP phase samples in the benchmarks. We then compare the measured time for the OpenMP phases to our predictions. Figure 3 shows the cumulative distribution of our prediction accuracy, i.e., the total percentage of OpenMP phases with error under the threshold indicated on the x-axis. The results demonstrate high accuracy of the model in all cases: more than 75% of the samples have less than 10% error.

V. Dynamic Concurrency Throttling

This section describes our two schemes ("profile-driven static mapping" and "one phase approach") for applying DCT. We predict performance for each OpenMP phase under all feasible concurrency configurations at the default frequency setting (highest frequency) with input from samples

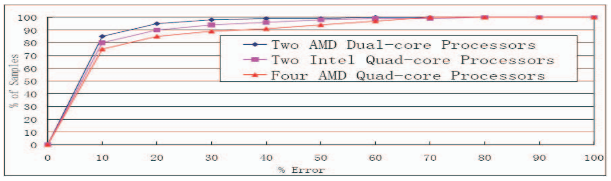


Figure 3: Cumulative distribution of prediction accuracy

of hardware event counters collected at runtime. We predict the execution time of each phase as discussed in Section IV. We cannot apply DCT in OpenMP phases where the code in each thread depends on the thread identifier, since this would violate correct execution. Also, we cannot make accurate time predictions for very short OpenMP phases due to the overhead of performing adaptation as well as accuracy limitations in performance counter measurements. We have empirically identified a threshold of one million cycles as the minimum DCT granularity for an OpenMP phase. We simply use the active configuration of the preceding phase for each phase below this threshold.

A. Profile-driven Static Mapping

Intuitively, using the best concurrency configuration for each OpenMP phase should minimize the computation time of each MPI task. We call this DCT strategy the *profile-driven static mapping*. To explore how well this strategy works in practice, we applied it to the AMG benchmark from the ASC Sequoia Benchmark suite. AMG has four OpenMP phases in the computation loop of its solve phase. Phases 1 and 2 are in phase group 1, phases 3 and 4 are in phase group 2, and the phase groups are separated by MPI_Waitall. We describe AMG in detail in Section VII. We run these experiments on two nodes, each with four AMD Opteron 8350 quad-core processors.

We first run the benchmark with input parameters P = [2 1 1], n = [512 512 512] under a fixed configuration throughout the execution of all OpenMP phases in all tasks for the entire duration of the run. We then manually select the best concurrency configuration based on these static observations, thus avoiding any prediction errors. Figure 4 shows the results with the fastest configuration for each task and OpenMP phase shown in stripes. Scalability varies across the phases and even within the same phase when executed in different tasks, due to differences in workload and data sets. The configuration of 4 processors and 2 threads per processor, shown as the first bar in each group of bars in Figure 5, has the lowest total time in the solve phase and, thus, is the best static mapping that we use as our baseline in the following discussion.

Under this whole-program configuration, each individual OpenMP phase may not use its best concurrency configuration. We select the best configuration for each OpenMP phase based on the results of Figure 4 and rerun the benchmark, as the second bar in each group of bars in Figure 5 shows. We profile each OpenMP phase with this

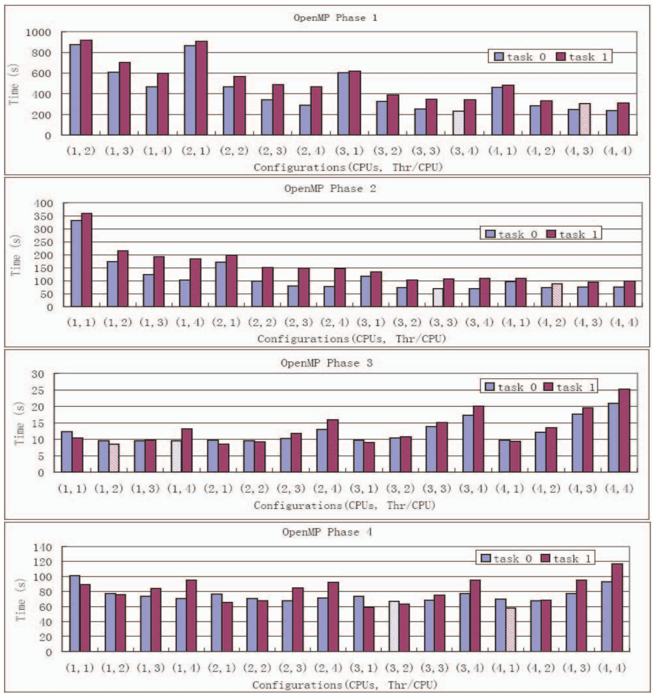


Figure 4: AMG phase performance

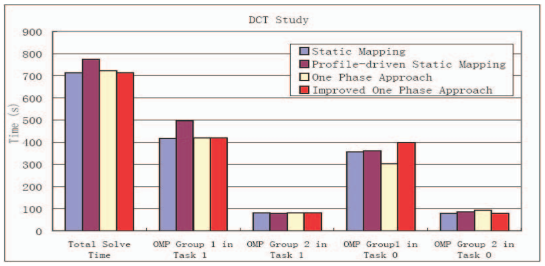


Figure 5: Impact of different DCT policies on AMG

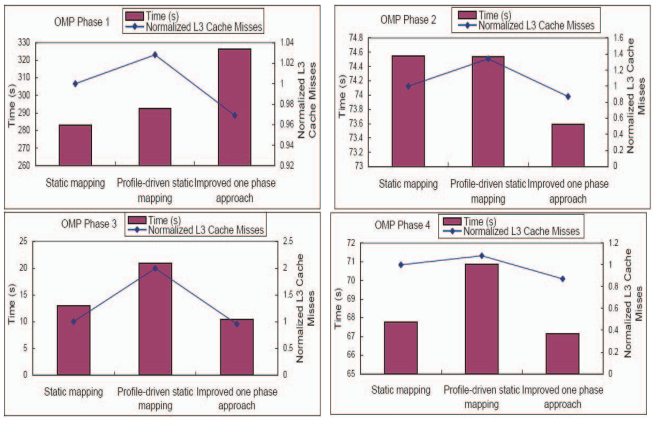


Figure 6: Phase profiles of task 0 under DCT policies

profile-driven static mapping, which we compare with the best static mapping to explore the source of the performance loss. Figure 6 shows the last-level cache misses that each OpenMP phase incurs normalized to the results with the static mapping. Three of the four OpenMP phases incur more misses with the profile-driven static mapping, leading to lower overall performance despite using the best configuration based on the fixed configuration runs. This increase arises from frequent configuration changes from one OpenMP phase to another under the profile-driven static mapping, thus confirming that cache interference causes the performance loss.

Previous work [9], [11] showed that the profile-driven static mapping can outperform the best static mapping. These results combine with ours to demonstrate that the profile-driven static mapping has no performance guarantees: it benefits from improved concurrency configurations while often suffering additional cache misses. We would have to extend our time prediction model to consider the configuration of the previous OpenMP phase in order to capture the impact on cache hit rates. We would also need to train our model under various thread mappings instead of a unique thread mapping throughout the run, which would significantly increase the overhead of our approach.

B. One Phase Approach

A simple solution to avoid cache misses caused by changing configurations is to use the same concurrency configuration for all OpenMP phases in each task in isolation. We can predict time for this combined phase and select the configuration that minimizes the time of the combined phase in future iterations under this *one phase approach*. Figure 5 shows that this strategy greatly reduces the performance loss for AMG compared to the profile-driven static mapping. Figure 6 shows that cache misses are also reduced significantly. However, we still incur significant performance loss compared to the best static mapping. Further analysis reveals that the one phase approach can change the critical task for specific phases despite minimizing the time across all OpenMP phases.

This problem arises because configurations are selected without coordination between tasks. Instead, each task greedily chooses the best configuration for each combined phase regardless of the global impact. Under our *improved one phase approach*, each task considers the time at the critical task when making its DCT decision. Each task selects a configuration that does not make its OpenMP phase groups longer than the corresponding ones in the critical task. Although this strategy may result in a configuration where performance for a task is worse than the one achieved with the best static mapping, it maintains performance as long as the OpenMP phase group time is shorter than the corresponding one in the critical task. Unlike the profile-driven static mapping, this strategy has a performance guarantee: it selects configurations that yield overall performance no worse than the best static mapping, as Figure 5 shows.

The profile-driven static mapping adjusts configurations at a fine granularity and suffers from the performance impact of cache interference between adjacent OpenMP phases. The one phase approach throttles concurrency at the coarsest granularity, thus ignoring that particular OpenMP phases may miss opportunities to execute with better configurations. The improved one phase approach strives for a balance between the two approaches by introducing task coordination and considering performance at a medium granularity (OpenMP phase groups).

VI. DVFS Control for Energy Saving

We follow DCT with DVFS to exploit further energy saving opportunities during OpenMP phases. The CPU frequency setting should satisfy the constraints of Equations 5 and 6. We use two steps for DVFS control: (1) identifying and estimating the slack available for DVFS; and (2) picking the appropriate frequency for each OpenMP phase given the slack.

A. Slack Estimation

Ideally, we can compute slack from Equation 4 in our model. We can estimate the communication time in the model a priori, using a communication benchmark such as MPPtest [17]. In practice, however, several factors can cause inaccuracies in our computed slack estimates. First, our execution time predictions for OpenMP phases have error, as shown in Section IV, and hence impact the computed slack. Second, since the workload of OpenMP phases can vary between outer iterations of the computation, our sampled iterations may have a different workload from that in other iterations. We introduce an error tolerance, ϵ, that adjusts our computed slack to compensate for these inaccuracies, thus preventing reductions of the frequency beyond the actual slack, which is the maximum time we can disperse to the OpenMP phases by DVFS without incurring performance loss. We modify our slack model in Equation 4 to:

$$\Delta t_i^{slack} = (t_c - t_i - t_i^{comm\_send} - t_{dvfs})/(1+\epsilon) \qquad (9)$$

In most cases, Equation (9) works well with a low value for ϵ. However, even a large ϵ (> 0.3) can lead to excessive frequency reductions in a few cases, which in turn hurts performance. Our analysis found that the actual slack can be shorter than the computed slack due to communication phases. To explain how communication can affect slack, Figure 7 illustrates a scenario observed in the IRS benchmark from the ASC Sequoia Benchmark Suite. In this case, three tasks execute OpenMP phase group 2, non-blocking MPI communication, MPI_Waitall, OMP phase group 3, and MPI_Allreduce. Due to load imbalance, the tasks arrive at MPI_Allreduce at different times. Task 0, the most heavily loaded, arrives late. Task 2 arrives earlier and can disperse the slack that our model computes. However, point-to-point

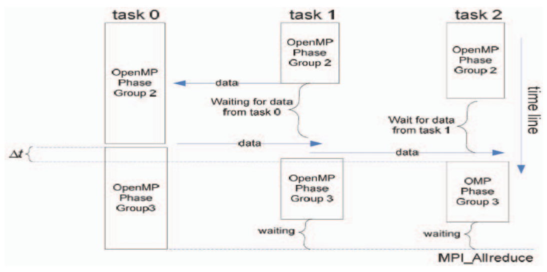


Figure 7: Impact of communication on slack with IRS

communication between group 2 and group 3, reduces the slack by Δt. In particular, task 2 starts executing OpenMP phase group 3 later than task 0, which reduces the available slack by Δt.

To capture the impact of communication on slack, we record the wait time in MPI operations (particularly MPI_Allreduce in Figure 7) and use it as an upper-bound of the slack we can disperse. The rationale is that any gap (Δt) will be reflected as a shortened wait time and our slack should never be longer than the wait time. If the computed slack is longer than the wait time, we simply disperse the wait time minus DVFS overhead. This heuristic enhances the accuracy of our prediction and decreases the effective ϵ value from 1.6 to 0.2 for IRS.

Simply using the wait time to estimate the slack is insufficient. The wait time, including communication time, has higher variance across iterations due to minor network perturbations. Thus, recorded wait time can lead to too high an estimate of slack. The slack computed from Equation (9), on the other hand, reflects the slack which is actually available in the tasks and therefore is a more reliable value that is often less than the actual wait time.

The selection of an appropriate ϵ depends on prediction accuracy and other factors as discussed above. According to our results (shown in Figure 3) and practical experiences, a value between 0.1 and 0.2 effectively compensates for errors while allowing energy-saving in most cases. We use $\epsilon \leq 0.2$ for our evaluation, which results in negligible performance loss. This value corresponds to prediction errors of 20% or less, which captures most of our results.

B. Selecting Frequencies

We choose an appropriate frequency for each OpenMP phase based on predictions of slack and computation time for each OpenMP phase under different frequency configurations. We adjust the frequency used for all phases that meet our time constraint (Equation 5) and minimize energy consumption (Equation 6).

We formulate the problem of selecting frequencies for OpenMP phases as a knapsack problem. Each OpenMP phase time under a particular frequency is an *item*. We associate a *weight* $w = \Delta t_{ijk}$ and a *value* $p = t_{ijk} f_k$ with each item. The weight is the time change under frequency f_k and the value captures relative energy. The total weight of all phases must be less than the slack, Δt_i^{slack}, and the total value of all phases should be minimized. Some items cannot be selected at the same time since we cannot select more than one frequency for each OpenMP phase. This is a variant of the 0-1 knapsack problem [18], which is NP-complete.

Dynamic programming can solve the knapsack problem in pseudo-polynomial time. If each item has a distinct value per unit of weight ($v = p/w$), the empirical complexity is $O((log(n))^2)$ where n is the number of items. We designed a unique dynamic programming solution to our problem. For convenience in its description, we replace $t_{ijk} f_k$ with $[(-1) \cdot t_{ijk} f_k]$ to solve the problem of maximizing the total value. Let L be the number of available CPU frequency levels, $w_{(i-1)*L+1}$, $w_{(i-1)*L+2}$, $\ldots$,w_{i*L} be the available weights of OpenMP phase i, and $p_{(i-1)*L+1}$, $p_{(i-1)*L+2}$, $\ldots$,p_{i*L} be the available values of OpenMP phase i. We denote the maximum attainable value with weight less than or equal to Y using items up to j as $A(j, Y)$, which we define recursively as:

$$A(0, Y) = -\infty, \qquad A(j, 0) = -\infty \tag{10}$$

$$\begin{aligned} &A(j, Y) = A(j - L, Y) + p_j, \\ &\text{if all } w_{j-1}, \ldots, w_{j-L+1} \text{are greater than Y.} \end{aligned} \tag{11}$$

$$\begin{aligned} &A(j,Y){=}\max\left(A(j{-}L,Y){+}p_j, \max_i(p_i{+}A(j{-}L,Y{-}w_i))\right), \\ &\text{for any } i \in [j - L + 1, j - 1], \text{ and } w_i \leq Y. \end{aligned} \tag{12}$$

We solve this problem by calculating $A(n, \Delta t_x^{slack})$ for task x, where n is the number of items. For a given total weight limitation W, the time complexity of this solution is linear in n.

VII. Performance Evaluation

We implemented our power-aware MPI/OpenMP system as a runtime library that performs online adaptation of DVFS and DCT. The runtime system predicts execution times of OpenMP phases based on collected hardware event rates and controls the execution of each OpenMP phase in terms of the number of threads, their placement on cores and the DVFS level. To use our library, we instrument applications with function calls around each adaptable OpenMP phase and selected MPI operations (collectives and MPI_Waitall). This instrumentation is straightforward and could easily be automated using a source code instrumentation tool, like OPARI [19], in combination with a PMPI wrapper library.

In this section, we evaluate our model with the Multi-Zone versions of NPB benchmarks (NPB-MZ) and two benchmarks (AMG and IRS) from the ASC Sequoia benchmark suite. The NPB-MZ [20] suite has three benchmarks (LU-MZ, SP-MZ and BT-MZ). Each has the same program flow, which Figure 8 shows. The benchmark loop has one procedure to exchange boundary values using point-to-point MPI communication. Therefore, the entire benchmark loop has only one OpenMP phase group. A bin-packing algorithm

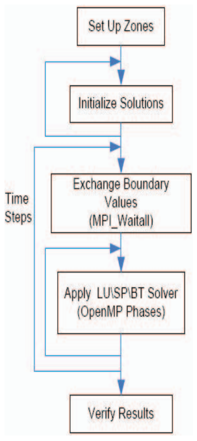
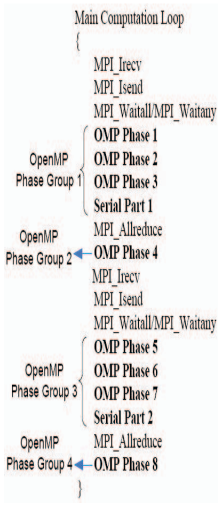
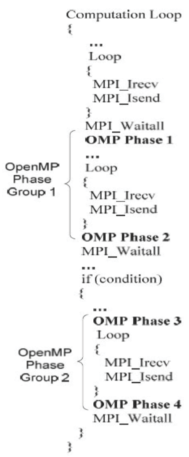

Set Up Zones
Initialize Solutions
Exchange Boundary Values (MPI_Waitall)
Apply LU\SP\BT Solver (OpenMP Phases)
Verify Results
Time Steps

Figure 8: NPB-MZ flow graph



```
Main Computation Loop
{
    MPI_Irecv
    MPI_Isend
    MPI_Waitall/MPI_Waitany
    OMP Phase 1
    OMP Phase 2
    OMP Phase 3
    Serial Part 1
    MPI_Allreduce
    OMP Phase 4
    MPI_Irecv
    MPI_Isend
    MPI_Waitall/MPI_Waitany
    OMP Phase 5
    OMP Phase 6
    OMP Phase 7
    Serial Part 2
    MPI_Allreduce
    OMP Phase 8
}
```

Figure 9: Simplified IRS flow graph



```
Computation Loop
{
  ...
  Loop
  {
   MPI_Irecv
   MPI_Isend
  }
  MPI_Waitall
  OMP Phase 1
  ...
  Loop
  {
   MPI_Irecv
   MPI_Isend
  }
  OMP Phase 2
  MPI_Waitall
  ...
  if (condition)
  {
   ...
   OMP Phase 3
   Loop
   {
    MPI_Irecv
    MPI_Isend
   }
   OMP Phase 4
   MPI_Waitall
  }
}
```

Figure 10: Simplified AMG flow graph

balances the workload of the OpenMP phases between all tasks. Under this algorithm, LU-MZ and SP-MZ allocate the same number of zones for each task and each zone has the same size. For BT-MZ, zones have different sizes and each task owns a different number of zones, however each task has almost the same total zone size. For our experiments, we introduce an artificial load imbalance in BT-MZ by modifying the load balancing code so that each task owns the same number of zones, but each task has a different total zone size. This load imbalance increases the energy saving opportunities of slack reclamation.

IRS uses a preconditioned conjugate gradient method for inverting a matrix equation. Figure 9 shows its simplified computational kernel. We group the OpenMP phases into four groups. Some OpenMP phase groups include serial code. We regard serial code as a special OpenMP phase with the number of threads fixed to 1. Although DCT is not applicable to serial code, it could be imbalanced between MPI tasks and hence provide opportunities for saving energy through DVFS. We use input parameters NDOMS=8 and NZONES_PER_DOM_SIDE=90. The IRS benchmark has load imbalance between the OpenMP phase groups of different tasks.

AMG [21] is a parallel algebraic multigrid solver for linear systems on unstructured grids. Its driver builds linear systems for various 3-dimensional problems; we choose a Laplace type problem (problem parameter set to 2). The driver generates a problem that is well balanced between tasks. We modified the driver to generate a problem with imbalanced load. The load distribution ratio between pairs of MPI tasks in this new version is 0.45:0.55.

We categorize hybrid MPI/OpenMP applications based on their OpenMP phases' workload characteristics: (1) imbalanced and constant workload per iteration (e.g., modified BT-MZ) or nearly constant workload per iteration (e.g., IRS); (2) imbalanced and non-constant workload per iteration (e.g., AMG); (3) balanced workload (e.g., SP-MZ and LU-MZ).

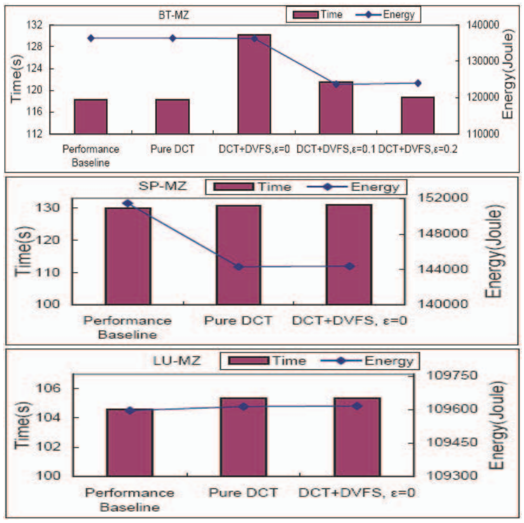


Figure 11: Adaptive DCT/DVFS control of NPB-MZ

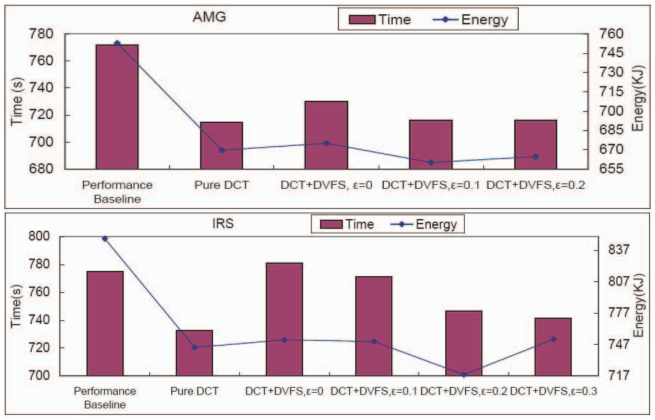


Figure 12: Adaptive DCT/DVFS control of AMG and IRS

We first run all benchmarks on two homogeneous nodes, each with four AMD Opteron 8350 quad-core processors (a total of 16 cores per node). The baseline is the execution under the configuration using 4 processors and 4 cores per processor, all running at the highest processor frequency. DVFS on the AMD Opteron 8350 has five frequency settings and we apply DVFS to the whole processor (all cores on a single socket). Figures 11 and 12 show the results.

Our DCT scheme selects the same concurrency configuration as the performance baseline for BT-MZ, which leads to no performance or energy gains. Due to good scalability of the OpenMP phases, the DCT strategy maintains maximum concurrency and cannot save energy. However after we apply DVFS, we achieve energy savings (10.21%) at $\epsilon = 0.2$ with no performance loss. When $\epsilon < 0.2$, we run each processor at a lower frequency but consume more energy due to increased execution time. The OpenMP phases in SP-MZ do not scale well, so we can save energy (5.72%) by applying DCT alone (pure DCT). Due to the balanced load in SP-

MZ, our DVFS algorithm cannot save energy, as shown by *Pure DCT* and *DCT+DVFS,* $\epsilon = 0$ having the same energy consumption. The LU-MZ benchmark has scalable OpenMP phases and balanced load and hence our runtime system does not have any opportunity to save energy. However, this test case shows that our system has negligible overhead (0.736%).

Our AMG problem has non-constant workload per iteration, which makes our predicted configurations based on sampled iterations incorrect in later iterations. After profiling its OpenMP phases, we find that AMG has a periodic workload. OpenMP phase group 1 has a period of 14 iterations and OpenMP phase group 2 has a period of 7 iterations. Thus, we can still apply our control schemes, but with application-specific sampling. Since the workload within a period varies from one iteration to another, we select configurations for every iteration within a period. We use more sample iterations during at least one period and change configurations for each iteration within a period. The results show that pure DCT achieves 11.56% energy saving and 7.39% performance gain. The best energy saving (13.80%) is achieved by applying DCT plus DVFS ($\epsilon = 0.1$) and the performance gain with DCT plus DVFS is 7.21%. In IRS, we observe 7.5% performance gain and 12.25% energy saving by applying only DCT. By applying DVFS, we can further reduce energy however with a slight performance loss, compared to the performance of pure DCT because the workload in OpenMP phases varies slightly and irregularly. The selection of our DVFS scheme based on sample iterations may hurt performance in the rest of the run. The best energy saving (13.31%) is achieved with combined DCT and DVFS with a performance loss of only 1.91%. We can reduce the performance loss by increasing ϵ, but this reduces energy saving while having limited performance gains.

To summarize, our hybrid MPI/OpenMP applications present different energy-saving opportunities and the energy-saving potential depends on workload characteristics. Our model can detect and leverage this potential. In particular, for imbalanced and constant (or close to constant) per iteration workloads, our algorithm is effective, saving energy while maintaining performance. For imbalanced and non-constant per iteration workload, if the workload is periodic, we can still apply our algorithm after detecting the periodicity of the workload; if the workload is totally irregular, our algorithm can fail. For balanced workloads, if OpenMP phases are non-scalable, we can save energy with pure DCT; if OpenMP phases are scalable, our algorithm does not save energy, but also does not hurt performance. We could detect the period of a workload by testing the applications with small problem sets. Alternatively, many scientific applications use a recursive computation kernel, thus creating a periodic workload that we could track based on the stack trace depth.

We extend our analysis into larger scales in order to investigate how our model reacts as the number of nodes changes. The following experiments consider the power awareness scalability of HPC applications, which we call the *scalability of energy saving opportunities*. We present results from experiments on the recently built System G supercomputer at Virginia Tech. System G is a unique research platform for Green HPC, with thousands of power and thermal sensors. System G has 320 nodes powered by Mac Pro computers, each with 2 quad-core Xeon processors. Each processor has two frequency settings for DVFS. The nodes are connected by Infiniband (40Gb/s). We vary the number of nodes and study how our power-aware model performs under strong and weak scaling. We use the execution under the configuration using 2 processors and 4 cores per processor and running at the highest processor frequency, which we refer to as (2,4), as the baseline by which we normalize reported times and energy.

Figure 13 displays the results of AMG and IRS under strong scaling input (i.e., maintaining the same total problem size across all scales). Actual execution time is shown above normalized execution time bars, to illustrate how the benchmark scales with the number of nodes. On our cluster, the OpenMP phases in AMG scale well, and hence DCT does not find energy-saving opportunities in almost all cases although, with 64 nodes or more, DCT leads to concurrency throttling on some nodes. However due to the small length of OpenMP phases at this scale, DCT does not lead to significant energy savings. When the number of nodes reaches 128, the per node workload in OpenMP phases is further reduced to a point where some phases become shorter than our DCT minimum phase granularity threshold and DCT simply ignores them. On the other hand, our DVFS strategy saves significant energy in most cases. However, as the number of nodes increases, the ratio of energy-saving decreases from 3.72% (4 nodes) to 0.121% (64 nodes) because the load difference between tasks becomes smaller as the number of nodes increases. With 128 nodes, load imbalance is actually less than DVFS overhead, so DVFS becomes ineffective. In IRS, our DCT strategy leads to significant energy-saving when the number of nodes is more than 8. We even observe performance gains by DCT when the number of nodes reaches 16. However DCT does not lead to energy-saving in the case of 128 nodes for similar reasons to AMG. DVFS leads to energy-saving with less than 16 nodes but does not provide benefits as the number of nodes becomes large and the imbalance becomes small.

Figures 14 displays the weak scaling results. We adjust the input parameters (AMG and IRS) or change the input problem definition (BT-MZ) as we vary the number of nodes so that the problem size per node remains constant (or close to it). For IRS and BT-MZ, the energy-saving ratio grows slightly as we increase the number of nodes (from 1.9% to 2.5% for IRS and from 5.21% to 6.8% for BT-MZ). Slightly increased imbalance, as we increase the problem size, allows additional energy savings. For AMG,

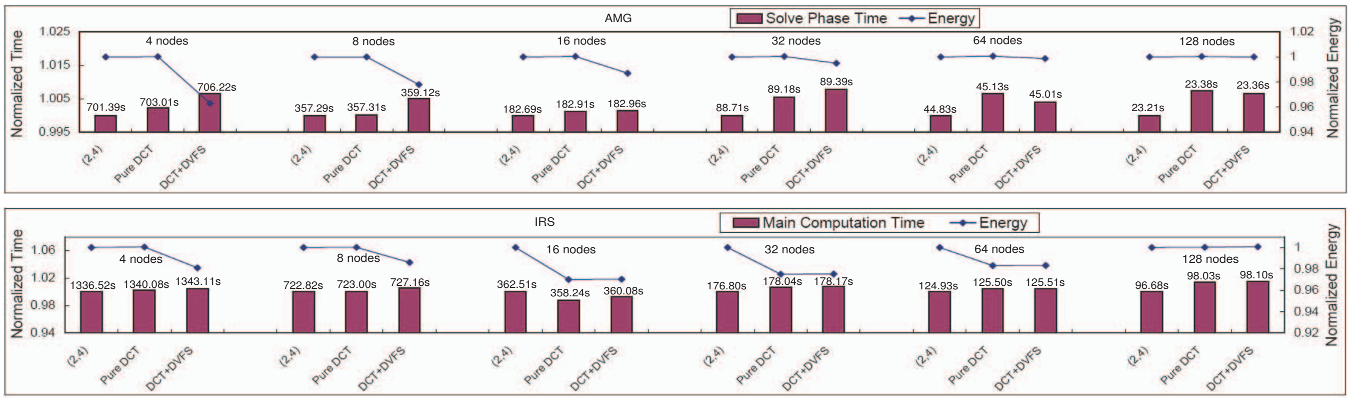


Figure 13: Results from strong scaling tests of our adaptive DCT/DVFS control on System G

we observe that the ratio of energy-saving stays almost constant (2.17%∼2.22%), which is consistent with AMG having good weak scaling. Since the workload per node is stable, energy saving opportunities are also stable as we vary the number of nodes.

In general, energy-saving opportunities vary with workload characteristics. They become smaller as the number of nodes increases under a fixed total problem size because the subdomain allocated to a single node becomes so small that the energy-saving potential that DVFS or DCT can leverage falls below the threshold that we can exploit. An interesting observation is that, when the number of nodes is below the threshold, some benchmarks (e.g., IRS with less than 16 nodes) present good scalability of energy saving opportunities for DCT because of the changes in their workload characteristics (e.g., scalability and working data sets) as the allocated sub-domain changes. With weak scaling, energy-saving opportunities are usually stable or increasing and actual energy-saving from our model tends to be higher than with strong scaling. Most importantly, under any case our model can leverage any energy saving opportunity without significant performance loss as the number of nodes changes.

VIII. Related Work

Several software-controlled techniques use DVFS to save energy in MPI programs. A heuristic by Freeh et al. [1] primarily attacks intra-node (memory) bottlenecks by choosing frequencies based on previously executed program phases. Kappiah et al. [8] address inter-node bottlenecks by using DVFS to exploit the *net slack* expected in an iteration. A scheduler that Springer et al. [4] propose selects node counts and CPU frequencies to minimize energy consumption and execution time. Rountree et al. [3] develop an offline method that uses linear programming to estimate the maximum energy saving possible for MPI programs based on critical path analysis. Subsequent work [22] provides a critical path-based online algorithm that uses simple predictions of execution times for program regions based on prior executions of the regions.

Our work differs from prior DVFS-based power management approaches in three ways. First, we choose CPU frequency configurations based on a scalable performance model instead of direct measurements or static analysis of slack time. Increasing numbers of processors, cores and available frequencies make scalable prediction models that prune the optimization space essential. Second, we consider hybrid MPI programs with nested OpenMP parallel phases that can be scaled using DVFS and DCT. Thus, the solution design space is more challenging although potential energy-efficiency improvements are also higher. Third, we consider systems with larger node counts and cores per node than earlier studies and, thus, derive insight into the implications of strong scaling, weak scaling, and multi-core processors for power management.

Curtis-Maury et al. study prediction models for adaptation via DCT and/or DVFS [9], [11], [16]. Their work targets pure OpenMP programs running on shared memory multi-core systems. They estimate performance for each OpenMP phase in terms of useful instructions per second for DVFS and DCT, which is sufficient within a shared memory node. We target hybrid MPI/OpenMP programs running on large-scale distributed systems and therefore must consider the implications of MPI communication on slack and the interactions between MPI communication events and OpenMP phases. Thus, our model must generalize their multi-dimensional prediction models for OpenMP phases and directly use the predicted time. We also address a shortcoming of their work, namely the lack of analysis of the implicit penalty of DCT on memory performance, which we analyze to design a new coordinated DCT algorithm that mitigates the penalty. Finally, we choose CPU frequencies under the constraints of both slack time and minimizing energy consumption instead of minimizing only execution time or only energy consumption.

Extensive prior research has explored optimization of power/thermal and performance properties of programs using feedback from hardware event counters. Isci et al. [7] and Merkel et al. [23] use hardware event counters to

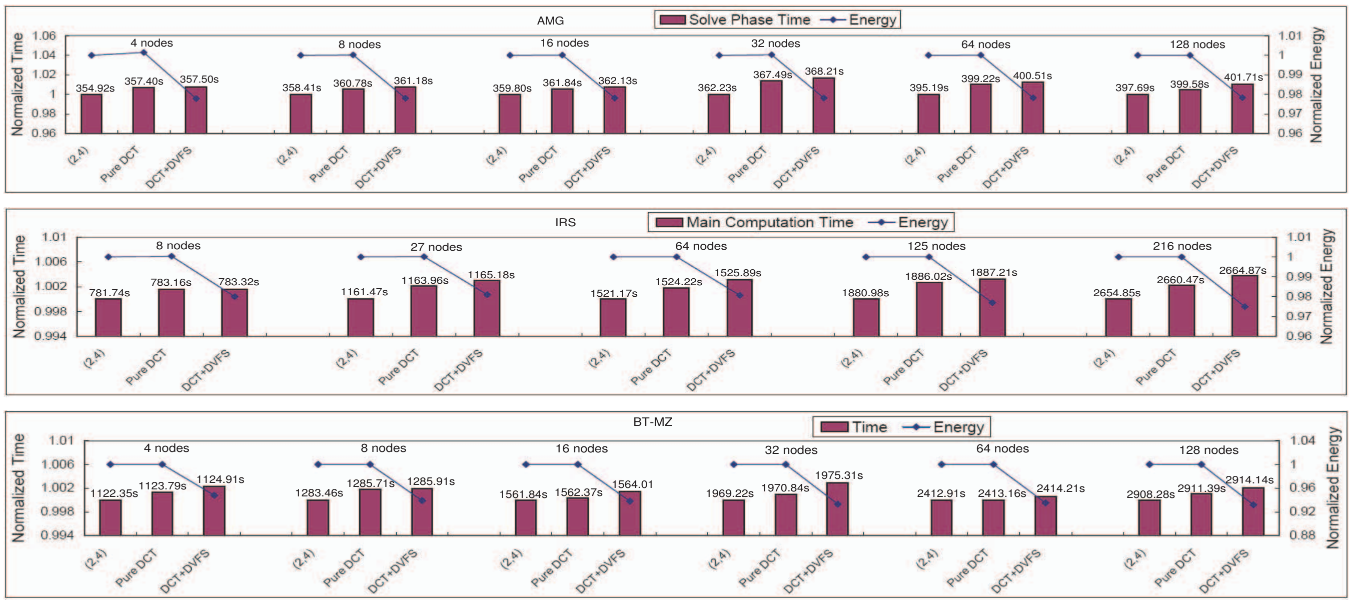


Figure 14: Results from weak scaling tests of our adaptive DCT/DVFS control on System G

determine the degree of utilization of each functional unit in a processor, from which they estimate power consumption or temperature. Based on these power and temperature estimates, they propose process scheduling algorithms. We use hardware event counters to capture statistical correlation between event samples and performance. By collecting specific counter events in sample iterations, our model learns program execution properties and makes accurate prediction for untested configurations thus reducing the design space for energy-efficiency optimization of large-scale, multicore systems.

IX. Conclusions

In this paper, we presented models and algorithms for energy-efficient execution of hybrid MPI/OpenMP applications and we characterized energy-saving opportunities in these applications, based on the interaction between communication and computation. We used this characterization, to propose algorithms using two energy-saving tools, DCT and DVFS, to leverage energy-saving opportunities without performance loss.

Our work improves existing DCT techniques by characterizing the potential performance loss due to concurrency adjustment. We use this insight to provide performance guarantees in our new "one phase approach", which balances between DCT performance penalties and energy savings. We also present a more accurate model for measuring slack time for DVFS control and solve the problem of frequency selection using dynamic programming. We apply our model and algorithm to realistic MPI/OpenMP benchmarks at larger scales than any previously published study. Overall, our new algorithm yields substantial energy savings (4.18% on average and up to 13.8%) with either negligible performance loss or performance gain (up to 7.2%). Further, our results are the first to characterize how energy saving opportunities vary under strong and weak scaling, on systems with large node and core counts.

In future work we intend to tune the accuracy of our prediction model. In particular, we will explore predictions that reflect the interference between concurrency-adjusted neighboring OpenMP phases. We will also include more factors that affect performance into our time prediction, which in turn should provide better guidelines for DCT and DVFS.

Acknowledgment

This work has been supported by NSF (CNS-0905187, CNS-0910784, CCF-0848670, CNS-0709025, CNS-0720750) and the European Commission through the MCF IRG project I-Cores (IRG-224759) and the HiPEAC Network of Excellence (IST-004408, IST-217068). Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-PROC-422990).

References

[1] V. Freeh and D. Lowenthal, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2007.

[2] R. Ge, X. Feng, and K. W. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," in *SC '05:*

Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 2005.

[3] B. Rountree, D. Lowenthal, S. Funk, V. Freeh, B. R. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," in *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007.

[4] R. Springer, D. Lowenthal, B. Rountree, and V. Freeh, "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2006.

[5] A. Miyoshi, C. Lefurgy, E. Hensbergen, R. Rajamony, and R. Rajkumar, "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling," in *Proc. of the International Conference on Supercomputing (ICS)*, 2002.

[6] C.-H. Hsu and W.-C. Feng, "A Power-Aware Run-Time System for High-Performance Computing," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[7] 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*, 2003.

[8] N. Kappiah, V. Freeh, and D. Lowenthal, "Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[9] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online Power-Performance Adaptation of Multithreaded Programs using Event-Based Prediction," in *Proc. of the 20th ACM International Conference on Supercomputing (ICS)*, 2006.

[10] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, "Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-Threaded Workloads on CMPs," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[11] 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 (TPDS)*, 2008.

[12] W. D. Gropp, "MPI and Hybrid Programming Models for Petascale Computing," in *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2008.

[13] R. Rabenseifner and G. Wellein, "Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures," *Int.J.High Perform. Comput. Appl.*, vol. 17, 2003.

[14] T.Mudge, "Power: A First-Class Architectural Design Constraint," *IEEE Computer*, vol. 34, no. 4, 2001.

[15] T. Horvath and K. Skadron, "Multi-Mode Energy Management for Multi-Tier Server Clusters," in *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[16] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores," in *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[17] W. Gropp and E. Lusk, "Reproducible Measurements of MPI Performance Characteristics," in *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 1999.

[18] M. Silvano and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[19] B. Mohr, A. D. Malony, S. Shende, and F. Wolf, "Design and Prototype of a Performance Tool Interface for OpenMP," in *Proceedings of LACSI 2001*, 2001.

[20] H. Jin and R. Van der Wijingaart, "Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks," in *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[21] V. E. Henson and U. M. Yang, "BoomerAMG: A Parallel Algebraic Multigrid Solver and Preconditioner," *Applied Numerical Mathematics*, vol. 41, 2000.

[22] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in *Proceedings of the 23rd International Conference on Supercomputing*, 2009.

[23] A. Merkel and F. Bellosa, "Task Activity Vectors: A New Metric for Temperature-Aware Scheduling," in *Third ACM SIGOPS EuroSys Conference*, 2008.