Strategies for Energy Efficient Resource Management of Hybrid Programming Models

Dong Li *Member, IEEE*, Bronis R. de Supinski, *Member, IEEE* Martin Schulz *Member, IEEE*, Dimitrios S. Nikolopoulos *Senior Member, IEEE*, Kirk W. Cameron *Member, IEEE*

Abstract—Many scientific applications are programmed using hybrid programming models that use both message-passing and shared-memory, due to the increasing prevalence of large-scale systems with multicore, multisocket nodes. Previous work has shown that energy efficiency can be improved using software-controlled execution schemes that consider both the programming model and the power-aware execution capabilities of the system. However, such approaches have focused on identifying optimal resource utilization for one programming model, either shared-memory or message-passing, in isolation. The potential solution space, thus the challenge, increases substantially when optimizing hybrid models since the possible resource configurations increase exponentially. Nonetheless, with the accelerating adoption of hybrid programming models, we increasingly need improved energy efficiency in hybrid parallel applications on large-scale systems. In this work, we present new software-controlled execution schemes that consider the effects of dynamic concurrency throttling (DCT) and dynamic voltage and frequency scaling (DVFS) in the context of hybrid programming models. Specifically, we present predictive models and novel algorithms based on statistical analysis that anticipate application power and time requirements under different concurrency and frequency configurations. We apply our models and methods to the NPB MZ benchmarks and selected applications from the ASC Sequoia codes. Overall, we achieve substantial energy savings (8.74% on average and up to 13.8%) with some performance gain (up to 7.5%) or negligible performance loss.

Index Terms—Power management, hybrid parallel programming models, dynamic concurrency throttling, dynamic voltage and frequency scaling.

1 Introduction

As core counts on processors increase exponentially, high-end computing systems built with multicore processors will provide unprecedented thread-level parallelism. To exploit multiple cores on a cache-coherent shared memory node, programmers tend to use programming models such as OpenMP [1]. At the same time, programmers tend to use message passing programming models, such as MPI [2], to achieve efficient execution of parallel applications on clusters of compute nodes with disjoint memories. With the trend towards high-end computing systems with tens of thousands of multicore, multisocket nodes, more programs will use *hybrid programming models*, such as MPI/OpenMP [3].

This accelerating adoption of hybrid programming models, will require improved resource management of hybrid applications. However, hybrid programming models impose great challenges on energy efficient resource management. The potential solution space increases substantially when optimizing hybrid models, since the possible resource configurations increase from one dimension (i.e., single programming model) to two dimensions (i.e., hybrid programming model). Further, different runtime systems implement each one aspect of these hybrid programming models. Lack of coordination between the runtimes and their contention for common hardware resources create inefficiencies. Modeling inefficiencies and then avoiding them is challenging.

Software-controlled power-aware execution of shared-memory applications leverages dynamic concurrency throttling (DCT) [4], [5] to provide energy savings. DCT controls the number of active threads that execute parallel regions to tune power and performance simultaneously [6]. Since the scalability of the regions can vary significantly due to system bottlenecks (e.g., memory bandwidth) or lack of concurrency, DCT often reduces both execution time and power consumption [6]. Alternatively, software-controlled power-aware execution can exploit different forms of slack to save energy. Slack can arise from an imbalanced workload between tasks, differences in the scalability of parallel workloads, different node capabilities, communication latency or any type of non-overlapped latency. Software-controlled power management algorithms leverage slack by dilating computation into slack intervals. Many state-of-the-art algorithms dilate computation into slack that occurs between MPI communication events, using dynamic voltage and frequency scaling (DVFS) [7], [8], [9], [10]. Similar algorithms can exploit slack in OpenMP, at barriers or other global synchronization points [11].

- *Dong Li is with Oak Ridge National Lab and Virginia Tech. Kirk W. Cameron is with the Department of Computer Science, Virginia Tech. Email: {lid, cameron}@cs.vt.edu*
- *Bronis R. de Supinski and Martin Schulz are with Lawrence Livermore National Lab. Email: {bronis, schulzm}@llnl.gov*
- *Dimitrios S. Nikolopoulos is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University of Belfast and nthe Foundation for Research and Technology – Hellas, Institute of Computer Science (FORTH–ICS). Email: dsn@ics.forth.gr*

In this paper, we integrate DCT and DVFS to provide energy-efficient resource management for hybrid MPI/OpenMP programs. Since the runtime systems are uncoordinated and the parameter space for energy saving is large, we must redesign DCT and DVFS based algorithms to consider the programming paradigm and the power-aware execution capabilities of the system. We must also carefully design strategies for applying these power-saving techniques, so that energy saving opportunities can be exposed and better leveraged. We make the following contributions:

- A model of hybrid MPI/OpenMP execution under varying degrees of shared-memory concurrency and waiting interval frequencies;
- The design of three DCT+DVFS strategies;
- An algorithm that aggregates OpenMP phases to reduce the implicit penalty of DCT on last-level cache misses and a novel DCT coordination scheme that mitigates the impact of throttling thread concurrency across MPI tasks in hybrid programs;
- A power estimation method that predicts system power consumption for workloads with variant concurrency and frequency configurations;
- A method to classify hybrid applications in order to facilitate the identification of slack time and characterizes the slack available for DVFS, considering both intra-node and inter-node interactions.
- A study of energy saving opportunities in both strong scaling and weak scaling using realistic parallel applications on system scales of up to 1024 cores.

The rest of this paper is organized as follows. Section 2 introduces the hybrid MPI/OpenMP programming model. We discuss DCT and DVFS in Section 3. Section 4 presents our combined DCT/DVFS strategies while Section 5 presents their methods to predict execution time and power. We provide implementation details in Section 6 and performance results in Section 7.

2 BACKGROUND

The hybrid MPI/OpenMP programming model exploits parallelism at the MPI task level and at the OpenMP loop level. Its hierarchical decomposition closely matches most large-scale HPC systems, which are clusters of multicore, multisocket nodes. We consider programs that use the THREAD_MASTERONLY model [3] in which a master thread invokes all MPI communication outside of OpenMP parallel regions. Almost all MPI implementations support the THREAD_MASTERONLY model.

Iterative parallel computations dominate execution time in scientific applications. Hybrid programming models exploit this iterative structure. Figure 1 depicts an iterative hybrid MPI/OpenMP computation that partitions the data space into subdomains, each handled by an MPI task. The MPI communication phase exchanges subdomain boundary data or computation results between tasks. OpenMP computation phases, or more simply *OpenMP phases*, follow the communication phase.

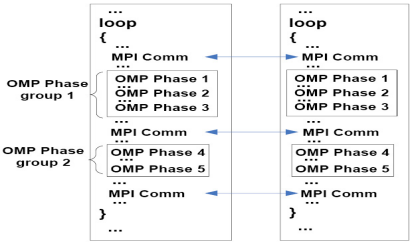


Fig. 1: Simplified typical MPI/OpenMP scheme

While, realistic hybrid MPI/OpenMP applications, such as AMG and IRS used in Section 7, can be far more complicated than Figure 1, we can abstract their program flows and simplify them to match that scheme.

MPI operations delineate multiple OpenMP phases with no MPI operations into *OpenMP phase groups*, as Figure 1 shows. Typically, MPI collective operations (e.g., MPI_Allreduce and MPI_Barrier) or grouped point-to-point completions (e.g., MPI_Waitall) separate these groups. MPI operations can incur slack since the wait times of different tasks can vary due to load imbalance. The *critical task*, which relates to critical path analysis, is the task upon which all other tasks wait.

Our energy saving methods dynamically adjust *configurations* of OpenMP phases of hybrid MPI/OpenMP applications. 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. We use OpenMP mechanisms to control the number of threads. We set the CPU affinity of threads using system calls. We select configurations so as to avoid performance loss while saving energy. We use execution time, energy and power, instead of the energy-delay product, to present results and to investigate any performance impact due to configuration selection. In the following sections, we first investigate the two energy saving techniques that we consider (DCT, DVFS) in isolation and then discuss strategies to apply them together.

3 APPLYING POWER SAVING TECHNIQUES

Both DCT and DVFS are common and effective: they can be implemented in user-level software with only rudimentary support from the operating system; they can be optimized to adapt to application characteristics dynamically; and they are suitable for high performance computing applications that often exhibit waiting periods due to synchronization, communication, or memory latency [4], [10], [8], [6], [12], [13], [14], [15], [16], [17]. Though widely studied, we must revisit DCT and DVFS strategies for use in hybrid programming models, a topic that we explore further in this section.

3.1 Applying DCT

DCT can reduce dynamic power consumption by putting cores that do not improve application execution time in

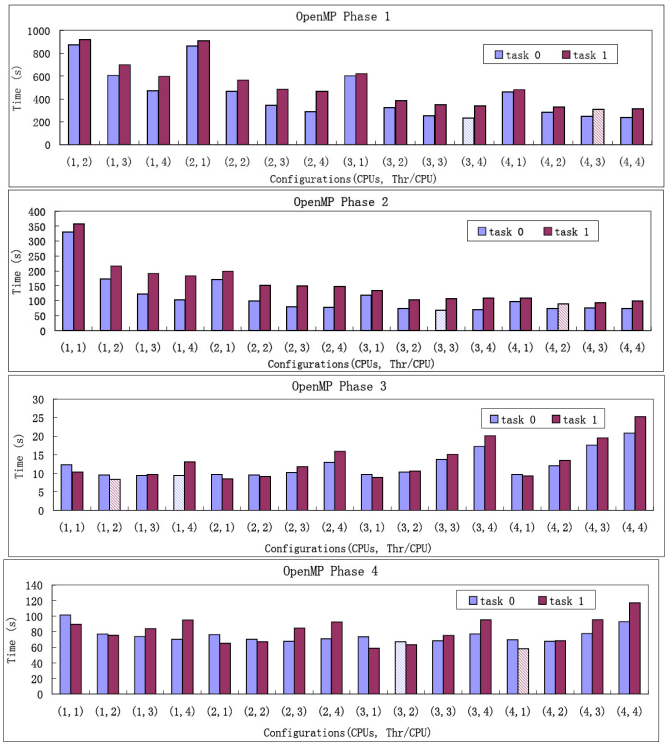


Fig. 2: AMG phase performance

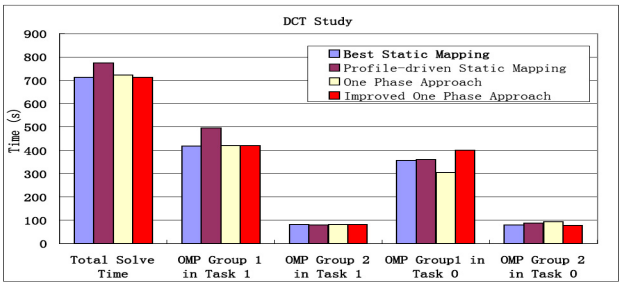


Fig. 3: Impact of different DCT policies on AMG

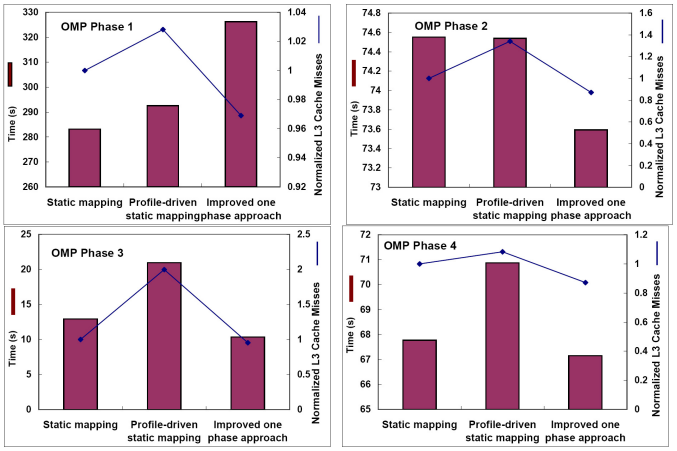


Fig. 4: Phase profiles of task 0 under DCT policies

a low-power state. DCT can also save execution time by alleviating contention for shared resources. Uncoordinated use of DCT for OpenMP phases of multiple MPI tasks may introduce load imbalance between tasks, thus creating more slack than the application would otherwise exhibit due to load imbalance at communication waiting points. In this section, we study how to apply DCT effectively in hybrid MPI/OpenMP applications.

To guide the discussion, we run the AMG benchmark from the ASC Sequoia Benchmark suite [18] on two nodes, each of which has four AMD Opteron 8350 quad-core processors. We report mean values for five trials of each test. An `MPI_Waitall` separates the two groups of AMG's solve phase. Each group consists of two OpenMP phases. We describe AMG in detail in Section 7. We run the benchmark with input parameters P=[2 1 1], n=[512 512 512] with two tasks. Figure 2 shows the results of profiling the OpenMP phases under different concurrency configurations; stripes indicate the fastest configuration for each task and OpenMP phase. Scalability varies within each MPI task across phases and within each phase across MPI tasks, due to workload differences between phases and tasks.

We profile an entire AMG run under each configuration to determine our baseline, the *best static mapping*, which is the mapping that yields the lowest execution time, when applied throughout program execution for all program phases. The best static mapping, which is the first bar in each group in Figure 3, uses 4 processors and 2 threads per processor on each node.

3.1.1 Profile-Driven Static Mapping

The best static mapping may not use the best concurrency configurations for individual OpenMP phases. For example, AMG Phases 3 and 4 do not achieve their best performance with the best static mapping. Intuitively, the *profile-driven static mapping*, which uses the best concurrency configuration for each individual OpenMP phase, should minimize the computation time of each MPI task.

We use the results of Figure 2 to determine the profile-driven static mapping. The second bar in each group in Figure 3 shows the execution time is longer overall than that of the best static mapping. We profile each OpenMP phase to determine the source of this performance loss. Figure 4 shows the results. Three of the four OpenMP phases lose performance with the profile-driven static mapping despite using the best concurrency configuration based on fixed configuration runs. This loss arises from the configuration changes, which lead to additional cache misses due to migration of data between caches on different cores. Last-level cache misses, which the lines in the charts on Figure 4 show normalized to the number with the best static mapping, are higher under profile-driven static mapping for all OpenMP phases.

Previous work [4], [6] has shown that the profile-driven static mapping can outperform the best static mapping. Overall, we observe that the profile-driven static mapping has no performance guarantees: it benefits from improved concurrency configurations but can suffer from additional cache misses due to migration of

data between caches on different cores. We next propose a scheme to address this problem.

3.1.2 One Phase Approach

A simple solution to avoid cache misses due to configuration changes, the *one phase approach*, combines all OpenMP phases in a task. We can then select the configuration that minimizes the time of the combined phase in future iterations. Under this DCT scheme, all OpenMP phases use the same concurrency configuration. We apply this strategy to AMG and, as Figure 3 shows, it greatly reduces the performance loss compared to the profile-driven static mapping. The one phase approach reduces cache misses but still incurs performance loss (4.8% in the computation phases) compared to the best static mapping. Further performance analysis reveals that the one phase approach uses suboptimal configurations for OpenMP phase groups despite minimizing the time across all OpenMP phases.

Our AMG results further illuminate this observation. Under the best static mapping (configuration (4,2)), the time in OpenMP phase group 2 is shorter in task 0 than task 1, as Figure 3 shows. Thus, task 1 is the critical task for phase group 2. Under the one phase approach, task 1 uses configuration (4,2) and achieves the same performance as the best static mapping, while task 0 uses the configuration (3,4), under which group 2 takes 10.46s longer than in task 1. Thus, task 0 becomes the critical task for group 2, which increases total computation time.

This problem arises because the one phase approach selects configurations without coordinating between tasks. Instead, each task greedily chooses the best configuration for the combined phase regardless of the global impact. Our *improved one phase approach* considers the critical task time when making DCT decisions. It selects a configuration for each task that does not make its OpenMP phase groups longer than those in the critical task. Although this strategy may select a configuration with lower performance than the best static mapping for a specific task (e.g., OpenMP phase group 1 in task 0, as Figure 3 shows), it maintains overall performance by keeping the OpenMP phase group time shorter than that of 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 3 shows.

In summary, fine granularity configuration changes under the profile-driven static mapping can increase cache misses. The one phase approach throttles concurrency at the coarsest granularity, thus ignoring the impact on individual OpenMP phases. The improved one phase approach strives for a balance between the two approaches through task coordination that considers performance at a medium granularity (phase groups).

3.2 Applying DVFS

A parallel application's MPI tasks can have different execution times because (1) the workload may not divide evenly between them; (2) their individual workload may scale differently; (3) the computing environment may be heterogeneous; or (4) the existence of distortion due to other system events, such as management of parallelism in the runtime system or operating system noise. Any of these events leads to load imbalance, which in turn causes slack in one or more MPI tasks. We use DVFS to reduce slack by extending the execution times of OpenMP phases of non-critical tasks, which in turn reduces overall energy consumption. We select frequencies for the cores that execute each OpenMP phase. For simplicity and to avoid introducing more load imbalance, we assume that all cores that execute an OpenMP phase within an MPI task use the same frequency.

We formulate frequency selection as a variant of the 0-1 knapsack problem [19], which is NP-complete. We define the time of each OpenMP phase under a particular core frequency f_k as an *item*. With each item, we associate a weight, w, that is the time change under frequency f_k compared to using the peak frequency and a value, p, that is the energy consumption under frequency f_k. The optimization objective is to keep the total weight of all phases under a given limit, which corresponds to the slack time Δt^{slack}, and minimize the total value of all phases. Our formulation is a variant of the basic problem since we require that some items cannot be selected together since we assume that we cannot select more than one frequency for each OpenMP phase.

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 [19]. For convenience in describing the dynamic programming solution of our variant, we replace p with $-p$ so that we maximize the total value. Let L be the number of available CPU frequency levels and, for OpenMP phase i ($0 \leq i < N$, where N is the number of phases), let $w_{i,j}$ ($1 \leq j \leq L$) be the lowest to highest available weights associated with CPU frequencies. Let $p_{i,j}$ ($1 \leq j \leq L$) be the available values. The total number of items is $n = N * L$. The total weight limit is W, the available slack time. The maximum attainable value with weight less than or equal to Y using items up to j is $A(j, Y)$, which we define recursively as:

$$A(0, Y) = -\infty, \qquad A(j, 0) = -\infty \tag{1}$$

For OpenMP phase i, if $\forall\ 1 \leq j < L$: $w_{i,j} > Y$, then

$$A(iL, Y) = A((i-1)L, Y) + p_{i,L}; \tag{2}$$

else,

$$\begin{aligned} A(iL, Y) = \quad & max(A((i-1)L, Y) + p_{i,L}, \\ & \max_j(p_{i,j} + A((i-1)L, Y - w_{i,j}))) \end{aligned} \tag{3}$$

We choose frequencies by calculating $A(n, \Delta t_i^{slack})$ for task i. For a given total weight limit W, the empirical complexity is still $O((log(n))^2)$. The supplementary material of this paper has a further discussion.

M	Number of OpenMP phases in a phase group
N	Number of MPI tasks
t_i	Time for OpenMP phases in task i
C	Set of possible concurrency configurations
t_{ijc}	Time for OpenMP phase j of task i under concurrency configuration c
t_{crit}	Critical time
t_{dvfs}	DVFS overhead
$t_i^{comm_send}$	Time to send data from task i
t_{ijf}	Time for OpenMP phase j of task i under frequency setting f
P_{ijf}	Average system power for OpenMP phase j of task i under frequency setting f
f^{max}, f^{min}	Maximum and minimum frequency settings
c^{max}	Maximal concurrency configuration
P_{ijc}	Average system power for OpenMP phase j of task i under concurrency configuration c
t_{ijcf}	Time for phase j of task i under concurrency configuration c and frequency f
P_{ijcf}	Average system power for phase j of task i under configuration c and frequency f

TABLE 1: Power-aware MPI/OpenMP model notation

4 DCT And DVFS Adaptation Strategies

We can apply DCT and DVFS in order or simultaneously to obtain energy savings. We now discuss the alternatives for combining DCT with DVFS in more detail and formalize the execution behaviors and energy saving opportunities. Table 1 summarizes the notation used in our formalization.

4.1 DCT First

We model this strategy, which first apples our improved one phase DCT approach in each MPI task, as:

$$t_i = \min_{c \in C} \sum_{j=1}^{M} t_{ijc} \tag{4}$$

where c is the concurrency configuration for task i and C is the set of all possible configurations on one node.

The critical task has the longest execution time in the phase group. We model this *critical time* as:

$$t_{crit} = \max_{1 \le i \le N} (\min_{c \in C} \sum_{j=1}^{M} t_{ijc}) \tag{5}$$

We compute the slack for task i (Δt_i^{slack}) with Equation 6. As Figure 5 shows, we reduce the available slack by the DVFS overhead and the time to send data from task i in order to avoid reducing the frequency too much.

$$\Delta t_i^{slack} = t_{crit} - t_i - t_i^{comm_send} - t_{dvfs} \tag{6}$$

After applying DCT to determine the concurrency configurations, we use communication to provide each non-critical task with the available slack. We then apply DVFS in these tasks so that the selected frequency satisfies our time constraint (Equation 7), which ensures that the total time change on task i does not exceed the available slack. Δt_{ijf} is the change in time between executing phase j on task i at frequency f and at the maximum frequency setting f_{max}, which we use as the DCT default.

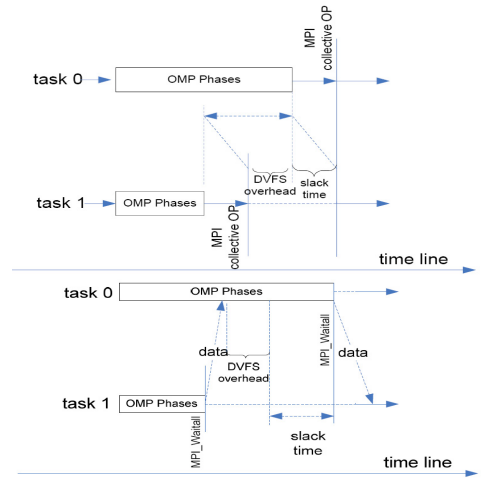


Fig. 5: Leveraging slack to save energy with DVFS

$$\sum_{1 \le j \le M} \Delta t_{ijf} \le \Delta t_i^{slack} \tag{7}$$

We also constraint DVFS energy consumption with the selected frequencies by the energy consumption at the maximum frequency (Equation 8).

$$\sum_{1 \le j \le M} P_{ijf} t_{ijf} \le \sum_{1 \le j \le M} P_{ijf^{max}} t_{ijf^{max}} \tag{8}$$

Under our time and energy constraints, we minimize the total energy consumption (E_i) with DVFS:

$$E_i = \min_{f^{min} \le f \le f^{max}} \sum_{1 \le j \le M} P_{ijf} t_{ijf} \tag{9}$$

4.2 DVFS First

This strategy uses communication to provide each non-critical task with the available slack before we apply DCT. We use Equation 6 to compute the slack. Following the requirements of Equations 7–9, we then apply DVFS. At the same time, the critical task performs DCT and communicates the new slack time to the non-critical tasks, which then perform DCT using the new slack, following the constraints of Equations 10–12, in which c^{max} is the maximal concurrency configuration (i.e., a thread running on every core of the node). These constraints are similar to Equations 7–9 but consider concurrency configurations instead of frequency settings.

$$\sum_{1 \le j \le M} \Delta t_{ijc} \le \Delta t_i^{slack} \tag{10}$$

$$\sum_{1 \le j \le M} P_{ijc} t_{ijc} \le \sum_{1 \le j \le M} P_{ijc^{max}} t_{ijc^{max}} \tag{11}$$

$$E_i = \min_{c \in C} \sum_{1 \le j \le M} P_{ijc} t_{ijc} \tag{12}$$

4.3 Simultaneous DCT and DVFS

With this strategy, we initially use communication to provide non-critical tasks with the available estimated slack. We then simultaneously apply DCT and DVFS in each task. We use additional communication to provide the non-critical tasks with the new slack, and proceed by simultaneously readjusting the DCT and DVFS settings of the non-critical tasks.

The critical task configurations follow Equation 13, which minimizes execution time. The configurations of the non-critical tasks follow Equations 14–16, which minimize energy consumption (Equation 16) subject to our new time (Equation 14) and energy (Equation 15) constraints. We compute the slack time from Equation 6.

$$t_i = \min_{c \in C, f^{min} \leq f \leq f^{max}} \sum_{j=1}^{M} t_{ijcf} \tag{13}$$

$$\sum_{1 \leq j \leq M} \Delta t_{ijcf} \leq \Delta t_i^{slack} \tag{14}$$

$$\sum_{1 \leq j \leq M} P_{ijcf} t_{ijcf} \leq \sum_{1 \leq j \leq M} P_{ijc^{max} f^{max}} t_{ijc^{max} f^{max}} \tag{15}$$

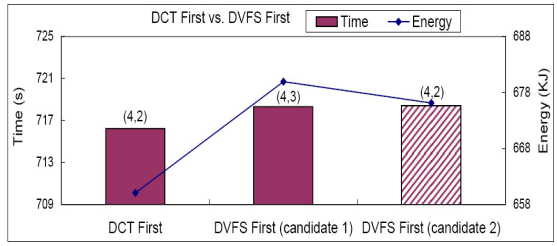
$$E_i = \min_{c \in C, f^{min} \leq f \leq f^{max}} \sum_{1 \leq j \leq M} P_{ijcf} t_{ijcf} \tag{16}$$

With Equations 9, 12, and 16, Equations 8, 11 and 15 are unnecessary. The minimization constraints (Equations 9, 12, and 16) guarantee that the selected configurations consume no more energy than with the maximum frequency and/or the maximum concurrency.

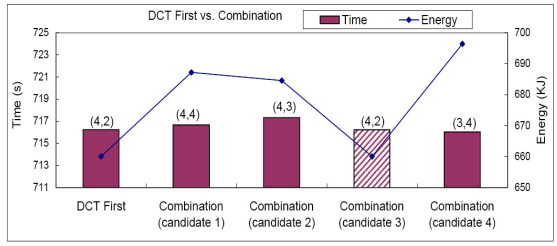
4.4 Performance

We perform tests with AMG to investigate how the three strategies affect energy saving and performance. We execute all tests on two nodes, each of which has four quad-core processors (AMD Opteron Processor 8350) with five available frequency settings. We measure the energy consumption of the nodes with a Watts Up Pro ES power meter. Except where specifically indicated, we use this type of power meter throughout the paper. To study how effectively DCT and DVFS contribute to energy savings, we display *candidate* configurations that satisfy the energy and time constraints but do not necessarily satisfy Equations 9, 12 and 16 (i.e., minimizing energy consumption). We mark the best candidate (i.e., the one with the lowest energy consumption) with stripes.

All tests improve performance (7.32% in average) over not using any power-saving technique (shown in Figure 13, but omitted here since we focus on comparing the power-saving techniques). The small performance differences (less than 0.5%) between tests mainly come from OS interference. Figure 6(a) compares the results of using DCT first or DVFS first. The numbers above the bars in the figure refer to the DCT configurations. For example, (4,2) uses four processors and two cores



(a)



(b)

Fig. 6: Comparison of three adaptation strategies

per processor. The results show that energy consumption is higher with the two DVFS candidates, which choose frequencies according to the slack before using DCT, than with DCT first. Candidate 2 consumes more energy despite using the same DCT configuration because it has a higher frequency setting. This slack is less than the slack after applying DCT under the DCT first strategy, so DVFS first chooses a higher frequency setting. Thus, candidate 2 does not fully leverage the available slack. Readjusting the frequency would be counter to the DVFS first design so we cannot fully leverage the slack to save energy. Candidate 1 consumes more energy than candidate 2 since it uses more processors and uses a DCT configuration that executes longer.

Figure 6(b) compares the results with the DCT first and combined strategies. These two strategies choose the same best configuration. Candidate 4 is interesting: its DCT configuration, (3,4), leads to slower execution than candidate 3's choice of (4,2), while its frequency setting is higher. In other words, to extend the execution time of the non-critical task to cover a fixed slack time, candidate 4 uses both DCT and DVFS, while candidate 3 only uses DVFS (DCT does not extend execution time). Since candidate 4 consumes more energy than candidate 3, we conclude that DVFS is more efficient in energy saving than DCT for this workload.

4.5 Discussion

Section 4.4 shows that the combined strategy saves the most energy. The results in Section 7 show that the combined strategy never performs worse than the other two strategies. If we regard the configuration space as a 2D plane with DCT configuration as one dimension and DVFS configuration as the other, DCT first (DVFS first) chooses a DCT (DVFS) configuration (i.e., chooses

a line in the 2D plane) and then chooses the frequency setting (concurrency setting) along this line. The best configuration may be a blind spot that neither DCT first or DVFS first can achieve. The combined strategy can choose any point, thus finding the blind spot.

Neither DCT first nor DVFS first guarantee the optimal configuration. DCT can increase or decrease slack. If slack increases, DCT first can save more energy by leveraging the additional slack. If slack decreases, the DVFS energy saving potential is reduced. DVFS first can save more energy than DCT first if DVFS is more effective than DCT for the workload.

5 Performance Prediction

Our strategies use execution time to compute slack (Equation 6) and to avoid performance loss (Equation 14). We also need execution time to compute energy consumption and to determine if we meet our energy constraints (Equation 15) and energy consumption is minimized (Equation 16). Further, we must estimate average system power (hereafter, simply power) to determine if a configuration meets our energy constraint (Equation 15). We also need power estimates to minimize energy consumption (Equation 16). In this section, we present our time and power prediction methods.

5.1 Time Prediction for OpenMP Phases

We present a method to predict execution time that extends previous work to predict IPC (Instructions Per Cycle) for DCT [4], [6], [13]. That work only required correct prediction of the rank ordering of DCT configurations for parallel execution phases and could tolerate inaccuracies in the absolute IPC predictions. We require accurate absolute predictions since we must estimate slack in order to apply our DCT and DVFS strategies.

Our prediction method uses execution samples to identify the execution time impact of configuration variations for each OpenMP phase. The sample phase input consists of elapsed CPU clock cycles and n hardware event rates ($e_{(1\cdots n,s)}$) observed with configuration s. An event rate is the number of event occurrences divided by the elapsed cycles. It captures the utilization of particular hardware resources that represent scalability bottlenecks, thus providing insight into the impact of hardware utilization and contention on scalability.

We use correlation analysis to choose model events. We select the n event rates that most strongly correlate with execution time (n is the number of available hardware performance counters, which is 2 or 4 on our platforms). The model predicts the time, $Time_t$, which is the time in OpenMP phases plus parallelization overhead, on a given target configuration (including both DCT and DVFS), t. For arbitrary samples, S, of size $|S|$, we model $Time_t$ as a linear function:

$$Time_t = \sum_{i=1}^{|S|}(Time_i \cdot \alpha_{(t,i)}(e_{(1\cdots n,i)})) + \lambda_t(e_{(1\cdots n,S)}) + \sigma_t \tag{17}$$

We define the term λ_t as:

$$\lambda_t(e_{(1\cdots n,S)}) = \sum_{i=1}^{n}(\sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,i,j,k)} \cdot e_{(i,j)} \cdot e_{(i,k)}))) + \sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,j,k,time)} \cdot Time_j \cdot Time_k)) + l_t \tag{18}$$

Equation (17) illustrates the dependency of terms $\alpha_{(t,i)}$, λ_t and σ_t on the target configuration. The term $\alpha_{(t,i)}$ scales the observed $Time_i$ on the sample configurations based on the observed values of the event rates in that configuration. The term λ_t combines the products of each event across configurations and of $Time_{j/k}$ to model interaction effects. The constant term σ_t is an event rate-independent term that includes parallel overhead. In summary, Equation (17) models each target configuration t through coefficients that capture the effects of hardware utilization at different degrees of concurrency, different mappings of threads to cores, and different frequency levels. Equation (18) defines the term λ_t. The term 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 to obtain the model coefficients (α, μ and constant terms) from training benchmarks, which we select empirically to vary properties such as scalability. We used MM5 and UA from the NAS OpenMP Parallel Benchmarks version 3.1 (119 total phases) as training benchmarks. The observed time $Time_i$, the product of $Time_i$ 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 model each target configuration separately.

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 (version 3.1) with CLASS B inputs using 247 phases in total. We collect event rates from three sample configurations (one using the maximum concurrency and frequency and the other two using configurations with half the concurrency and the second highest frequency) 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 7 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 good accuracy of the model in all cases: more than 75% of the samples have less than 10% error.

5.2 Power Estimation

Power is approximately linear in processor frequency and CPU utilization for relatively high utilization [20]. Since HPC applications typically exhibit high CPU utilization, we can estimate power based on frequency.

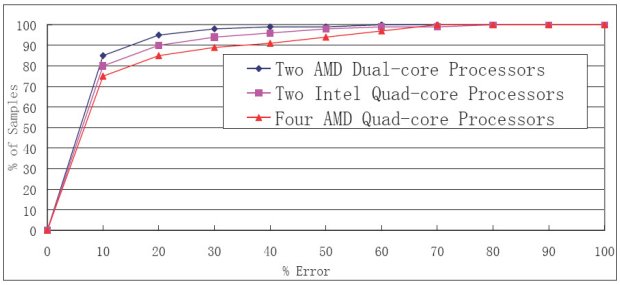


Fig. 7: Cumulative distribution of time prediction accuracy

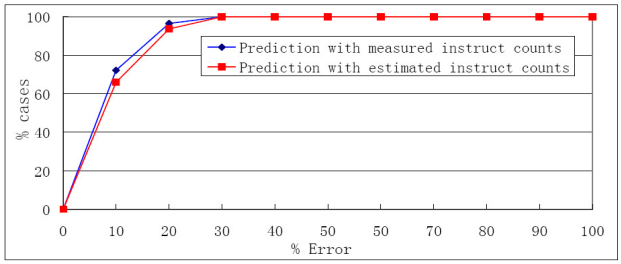


Fig. 8: Cumulative distribution of power prediction accuracy

Unfortunately, this method only works for our DCT first strategy in which all candidates use the same concurrency configuration. For our DVFS first and combined strategies, the candidates can use different concurrency configurations so their CPU utilization can vary significantly. Thus, we need a new power estimation method.

Previous work [21] found a strong correlation between power consumption and IPC if the processor frequency is fixed. When the frequency varies, the impact of processor stalls on IPC also varies. On lower frequencies, processor stalls are smaller components of consumed CPU clock cycles than on higher frequencies, which in turn tends to increase IPC on lower frequencies. Thus, IPC cannot reflect power when frequency changes. We instead use a linear model (Equation 19) to capture the relationship between instructions per second (IPS) and power consumption to estimate power. We obtain the model parameters (k_1, k_0) by offline training. Given the IPS of a configuration, we estimate the power at runtime as:

$$P = k_1 \times IPS + k_0 \tag{19}$$

To calculate IPS, we need the instruction count and execution time. We predict execution time as in Section 5.1. The instruction count is independent of processor frequency, but dependent on the number of threads that execute the OpenMP phases. Using more threads tends to increase the instruction count due to additional parallel overhead. We record instruction counts from sample configurations and use linear interpolation to estimate the counts for the other (untested) configurations. Interpolation constructs new data points *within* a range of known data points. Thus, the predicted value is bounded by known values, thus limiting prediction error. We use interpolation instead of the linear regression used in Section 5.1, since the instruction counts should be within the range of values collected from samples with minimum and maximum concurrency. Further, we choose *linear* interpolation since the instruction count intuitively varies across DCT configurations primarily due to data accesses and thread synchronization, which are approximately *linear* in thread count.

We train our model with three benchmarks (bt.B, cg.C and ft.B) from the OpenMP version of the NAS Parallel Benchmarks with 239 total configurations. We predict the power of the other four NAS benchmarks (260 configurations) on a machine with four AMD Opteron 8350 quad-core processors. Figure 8 shows the cumulative accuracy: prediction with estimated instruction counts is close that with measured instruction counts. Our prediction scheme exhibits good accuracy: about 80% of the predictions have error less than 18% and more than 90% have error less than 25%.

Discussion: Our time and power predictions achieve good accuracy for most cases. However, in a few cases when prediction is inaccurate, performance loss may happen. We use 148 OpenMP phases from the NAS parallel benchmark suite to investigate further how our prediction accuracy can impact performance. Our results show that our prediction chooses the best configuration for 130 out of 148 cases (87.8%). Between the other 18 cases, only 8 incur performance loss. The remaining 10 cases do not achieve the best energy savings but maintain performance. To avoid performance loss, our algorithm uses one more iteration to determine if performance suffers under the chosen configuration. If it does, the system uses to the original configuration.

6 Implementation

We have implemented our power-aware strategies for MPI/OpenMP programs in a runtime system that adapts dynamically using DVFS and DCT. To enable the runtime system, we instrument applications with function calls around OpenMP phases and selected MPI operations (collectives and MPI_Waitall). This instrumentation is mechanical and can be automated with a source code instrumentation tool, such as OPARI [22], in combination with an MPI interposition library.

We cannot apply DCT to OpenMP phases in which the code in each thread depends on the thread identifier, since changing thread counts would violate correct execution. Also, we cannot accurately predict time and power for short OpenMP phases due to the overhead of performing adaptation as well as accuracy limitations in performance counter measurements. We empirically identify a threshold of one million cycles as the minimum DCT granularity for an OpenMP phase. For each phase below this threshold, we simply use the active configuration of the preceding phase.

Our model computes slack time from Equation 6. Details on how we compute the terms of the model are

provided in the supplementary material that accompanies the paper.

We collect hardware events from performance counters to learn application execution properties. We use four sample configurations: one uses the maximum concurrency and frequency and one uses the minimum concurrency and frequency to bound instruction counts for linear interpolation (Section 5.2), while the other two use configurations with half the concurrency – with different mappings of threads to cores – and the second highest frequency. This small number of samples provides insight into the utilization of shared caches and memory bandwidth. Based on sample events, we then predict execution time of the untested configurations with the coefficients obtained by offline training and our time prediction model (Section 5.1). After collecting hardware events, we use communication to determine the critical task and available slack time. Based on slack time and the adaptation strategy (Section 4), we choose the configuration candidate that satisfies the time constraint. We then choose the best configuration based on the power model (Section 5.2). The best configuration minimizes energy consumption while satisfying our energy constraint (Section 4). We enforce the configuration decisions with the Linux processor affinity system call, $sched_setaffinity()$, threading library-specific calls for changing concurrency levels ($omp_set_num_threads()$), and a set of $cpufreq$ pseudofiles in the $/sys$ directory for changing processor frequencies. The rest of the execution uses the predicted optimal configurations.

7 Performance Evaluation

We evaluate our model with the Multi-Zone versions of the NAS Parallel benchmarks (NPB-MZ) [23] and two full applications (AMG and IRS). Figure 9 shows the program flow of the three benchmarks (LU-MZ, SP-MZ and BT-MZ) of NPB-MZ [24]. The benchmark loop has one procedure to exchange boundary values using point-to-point MPI communication. Thus, the entire benchmark loop has only one OpenMP phase group. A bin-packing algorithm 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.

IRS uses a preconditioned conjugate gradient method for inverting a matrix equation. Figure 10 shows its simplified computational kernel. The OpenMP phases form four groups. Some OpenMP phase groups include serial code, which we treat as a special OpenMP phase with the number of threads fixed to 1. Although DCT is not applicable to serial code, serial code could be imbalanced between MPI tasks hence providing opportunities for saving energy through DVFS. We use input parameters NDOMS=8 and NZONES_PER_DOM_SIDE=90. The IRS

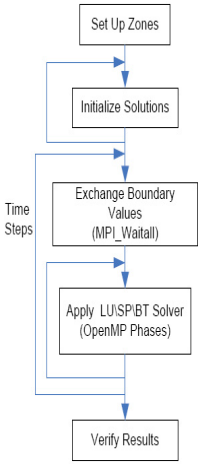


Fig. 9: NPB-MZ flow graph

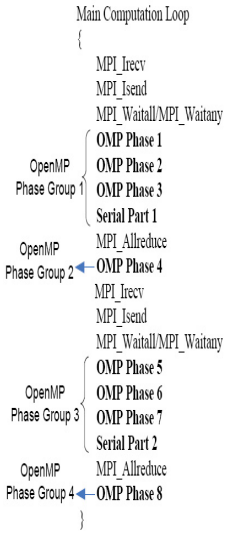


Fig. 10: Simplified IRS flow graph

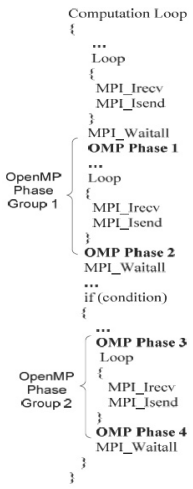


Fig. 11: Simplified AMG flow graph

benchmark has load imbalance between the OpenMP phase groups of different tasks.

AMG [25] 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.

For our experiments, we introduce artificial load imbalance into BT-MZ and AMG to investigate how the energy saving varies with explicit load imbalance under different strategies. However, our system does not necessarily require imbalanced load to save energy. We choose BT-MZ and AMG because we can easily change their problem partitions across tasks without violating the correctness of the applications. In particular, we slightly modify BT-MZ so that each task owns the same number of zones, but each task has a different total zone size. We also generate a new problem with imbalanced load for AMG. 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 the workload characteristics of their OpenMP phases: 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., modified AMG); 3) balanced workload (e.g., SP-MZ, LU-MZ, BT-MZ and AMG).

7.1 Basic Tests

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, running at the highest processor frequency. DVFS on the AMD Opteron 8350 has five frequency settings and we apply DVFS to the all cores

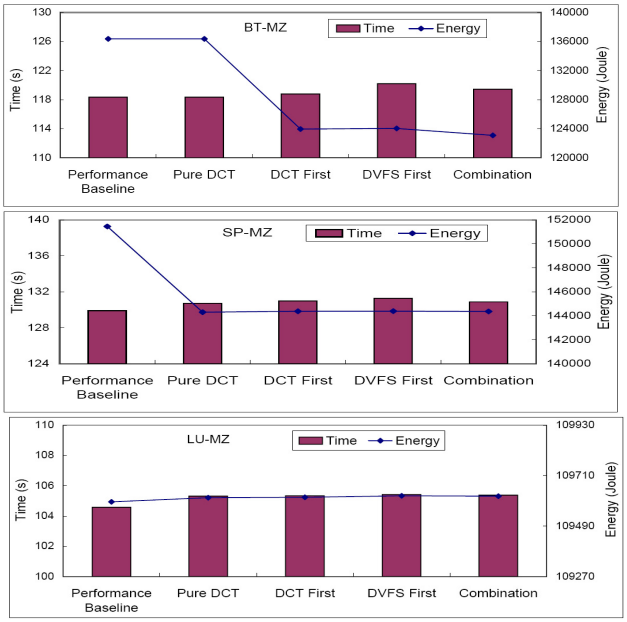


Fig. 12: Execution time and energy consumption of NPB-MZ

on a single socket. We display the results with three adaptation strategies (Section 4). The results with pure DCT are also displayed so that we can know whether DCT or DVFS lead to the energy savings.

Figure 12 shows the results for the NPB benchmarks. The pure DCT scheme selects the same concurrency configuration as the performance baseline for BT-MZ, which leads to no performance or energy gains. Due to the scalability of the OpenMP phases, DCT maintains maximum concurrency and cannot save energy. However with DCT first, we achieve energy savings (10.21%) with almost no performance loss. With DVFS first, we choose the same DCT/DVFS configurations as DCT first and thus the energy consumption is almost the same. With the combined strategy, we achieve the largest energy saving (10.85%), because it can choose a configuration that the other two strategies cannot.

The OpenMP phases in SP-MZ do not scale well, so we can save energy (5.72%) by applying DCT alone. Due to the balanced load in SP-MZ, our DVFS algorithm cannot save energy, as shown by pure DCT and the three strategies having the same energy consumption. The LU-MZ benchmark has scalable OpenMP phases and balanced load so our runtime system does not save energy. However, this test case shows that our system has negligible overhead (0.736%).

The AMG problem that we study 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. Therefore, we can still apply our control schemes, but with application-specific sampling. Since the workload within a period

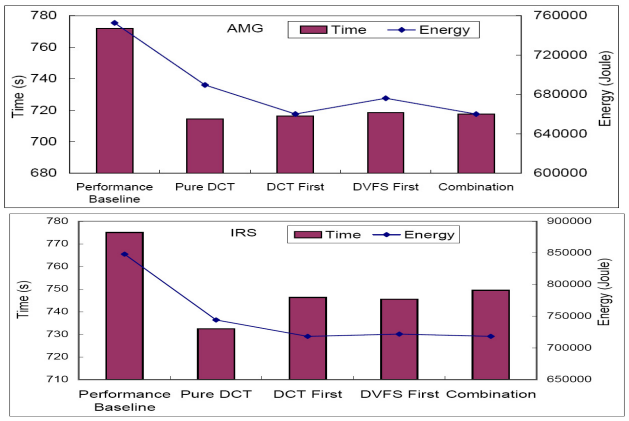


Fig. 13: Execution time and energy consumption of AMG and IRS

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 AMG results in Figure 13 show that pure DCT achieves 8.38% energy saving and 7.39% performance gain. The best energy saving (13.80%) is achieved by applying either DCT first or the combined strategy. DVFS first, however, achieves less energy saving since it cannot fully leverage slack time (discussed in Section 4.4). In IRS, we observe 7.5% performance gain and 12.25% energy saving by applying only DCT. Our three strategies to apply DVFS and DCT save additional energy although we incur 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 DCT first and the combined strategy, with overhead of only 1.89% and 2.07% respectively, compared to pure DCT.

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 balanced workloads, if OpenMP phases are non-scalable, we can save energy with DCT; if OpenMP phases are scalable, our algorithm does not save energy, but also does not hurt performance. 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 manually detecting the periodicity of the workload.

7.2 Scaling Tests

We extend our analysis to larger systems to investigate the scalability of our energy saving techniques. We present results from experiments on the System G

supercomputer at Virginia Tech. System G is a research platform for Green HPC, composed of 320 nodes powered by Mac Pro computers, each with 2 quad-core Xeon processors, and thousands of power and thermal sensors. Each processor has two frequency settings for DVFS. The nodes are connected by Infiniband (40Gb/s). System power is measured with Raritan Dominion PX power strips. 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 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 14 displays the results of AMG and IRS under strong scaling (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 so 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 save significant energy. When the number of nodes reaches 128, the per node workload in OpenMP phases is further reduced to a point at which some phases become shorter than our DCT minimum phase granularity threshold and DCT simply ignores them.

On the other hand, our three strategies using DVFS save 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. We also notice the slightly increased overhead as the number of nodes increases, which arises from an increase in the cost of communication to coordinate tasks under our strategies. In IRS, our strategies with DCT lead to measurable energy-saving when the number of nodes is more than 8 (up to 3.06%). 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 savings with less than 16 nodes but does not provide benefits as the number of nodes becomes large and the imbalance becomes small.

Figure 15 displays our weak scaling results. We adjust the input parameters as we vary the number of nodes so that the problem size per node remains constant (or close to it). For IRS, the energy saving ratio grows slightly as we increase the number of nodes (from 1.9% to 2.5%). Slightly increased imbalance, as we increase the problem size, allows additional energy savings. For AMG, we observe that the ratio of energy-saving stays almost constant (2.17%$\sim$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.

Energy saving opportunities generally vary with workload characteristics. They decrease as the node count increases under a fixed total problem size since the workload of a single node may fall below our exploitable threshold. With weak scaling, energy saving opportunities are usually stable or increasing and actual energy savings from our model tend to be higher than with strong scaling. Most importantly, our model always leverages energy saving opportunities without significant performance loss as the node count changes.

8 Related Work

OpenMP Performance Prediction: Curtis-Maury et al. study prediction models for adaptation via DCT and/or DVFS [4], [6], [13]. They estimate performance for each OpenMP phase in terms of useful IPS for DVFS and DCT, which is sufficient within a shared memory node. Since we target hybrid MPI/OpenMP programs running on large-scale distributed systems, we must consider the impact 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: the lack of analysis of the implicit penalty of DCT on memory performance. Our analysis leads to a new coordinated DCT algorithm that mitigates the penalty. Finally, we choose CPU frequencies under the constraints of both slack and minimizing energy consumption instead of minimizing only execution time or only energy consumption.

Liao et al. [26] proposed static analytical models to predict OpenMP performance with the support of the OpenUH compiler. Their models account for three performance factors: memory hierarchy; processor count; and parallel overhead. Tournavitis et al. [27] integrate static analysis with profiling information about control and data dependencies. They use a machine learning method to determine the performance benefits of OpenMP parallelism. However, these works do not consider *both* power consumption and execution time. They also focus on individual OpenMP phases, instead of holistically considering parallelism overhead.

Power-Aware MPI: Kappiah et al. [12] address internode bottlenecks by using DVFS to exploit the *net slack* expected in an iteration. Son et al. [28] leverage DVFS for both CPU and communication links in a coordinated fashion to save energy specifically for parallel sparse matrix applications. A scheduler that Springer et al. [14] propose selects node counts and CPU frequencies to minimize energy consumption and execution time. A heuristic by Freeh et al. [16] primarily attacks intranode (memory) bottlenecks by choosing frequencies based on previously executed program phases. Rountree et al. [15] develop an offline method that uses linear programming

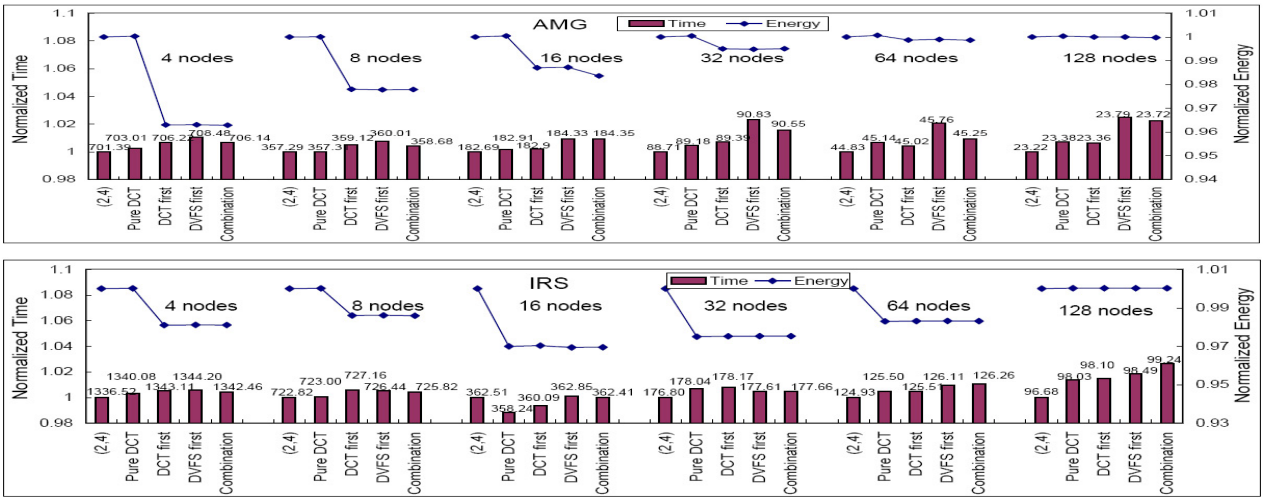


Fig. 14: Results from strong scaling tests of adaptive DCT/DVFS control on System G

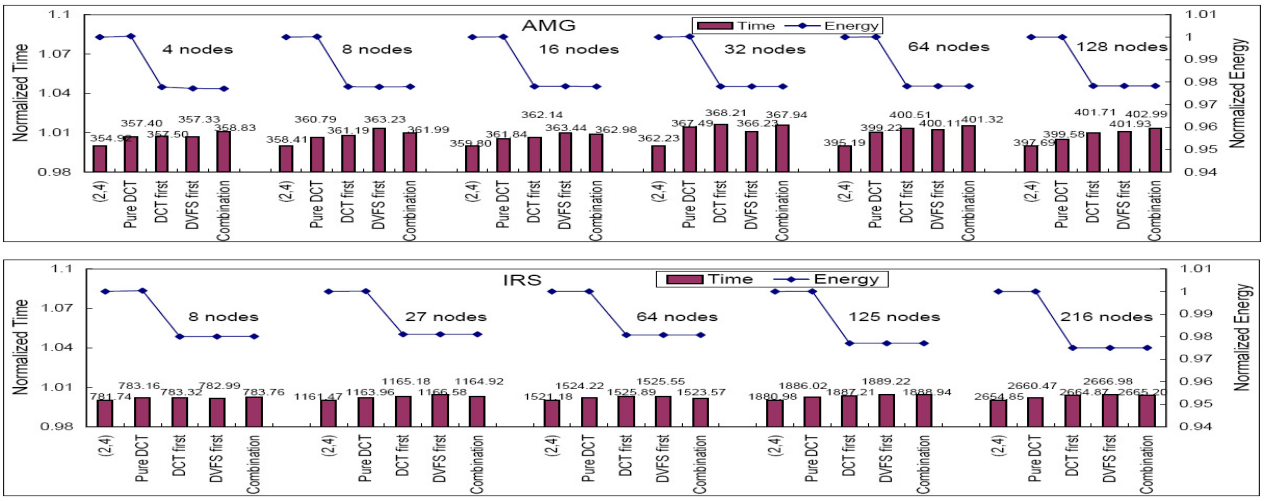


Fig. 15: Results from weak scaling tests of adaptive DCT/DVFS control on System G

to estimate the maximum energy saving possible for MPI programs based on critical path analysis. Subsequent work [17] 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 slack analysis. 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 savings are also higher. Third, we consider systems with larger node counts and cores per node and, thus, derive insight into the implications of strong scaling, weak scaling, and multicore processors for power management.

9 Conclusions

In this paper, we presented models and algorithms for energy efficient execution of hybrid MPI/OpenMP applications. We characterized energy saving opportunities in these applications, based on the interaction between communication and computation. We used this characterization to propose algorithms that use two energy saving tools, DCT and DVFS, to leverage energy saving opportunities without performance loss. We studied three strategies to apply DCT and DVFS and investigated their impact on energy savings. We proposed a power estimation model that can estimate power for variant concurrency and frequency configurations.

Our work improved existing DCT techniques by characterizing the potential performance loss due to concurrency adjustment. We used this insight to provide performance guarantees in our improved one phase approach, which balances DCT performance penalties and energy savings. We also presented a more accurate model for measuring slack time for DVFS control and

solved the problem of frequency selection using dynamic programming. We applied our model and algorithms to realistic MPI/OpenMP benchmarks at larger scales than any previously published study. Overall, our new algorithm yields substantial energy savings (8.74% on average and up to 13.8%) with either negligible performance loss or some performance gain (up to 7.5%). 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.

Our work provides new opportunities for research of concurrency throttling at the MPI level, an important topic that will involve substantial work to understand mechanisms for MPI task aggregation and their impact on computation and communication. For SPMD programs, the computation performance prediction may leverage our DCT performance prediction methodology. However, the communication phases within MPI tasks introduce significant complexity into the research. The communication phases across MPI tasks are usually different and have data dependencies. How to account for prediction inaccuracy caused by communication time differentiation across tasks and across different concurrency levels, how to group tasks to optimize communication performance, and how to predict the performance of MPI group communication primitives and optimized collective operations are major research challenges. In addition, applying DCT to message passing models requires complex on-the-fly data redistribution and process migration operations, which is another challenge.

10 Acknowledgments

This work has been partially supported by the European Commission under the I-CORES project (FP7 MCF-IRG Contract #224759) and by NSF (CNS-0905187, CNS-0910784, CCF-0848670, CNS-0709025, CNS-0720750). Portions of this work were performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DEAC52-07NA27344 (LLNL-JRNL-XXXXXX).

References

[1] OpenMP Architecture Review Board, "OpenMP Fortran/C/C++ Application Programming Interface," May 2008, version 3.0.

[2] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, 2nd ed. MIT Press, 1998, vol. 1.

[3] 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.

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

[5] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, "Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multithreaded Workloads on CMPs," in *Proc. 13th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, 2008, pp. 277–286.

[6] 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.

[7] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar, "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling," in *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, 2002, pp. 35–44.

[8] H. Chung-Hsing and F. Wu-Chun, "A Power-Aware Run-Time System for High-Performance Computing," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[9] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in *36th International Symposium on Microarchitecture*, 2003, pp. 93–104.

[10] V. Freeh, N. Kappiah, D. Lowenthal, and T. Bletsch, "Just-In-Time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI Programs," in *Proceedings of the 19th Annual ACM International Conference on Supercomputing*, 2005.

[11] Y. Dong, J. Chen, X. Yang, L. Deng, and X. Zhang, "Energy-Oriented OpenMP Parallel Loop Scheduling," in *International Symposium on Parallel and Distributed Processing with Applications*, 2008.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Lawrence Livermore National Laboratory, "ASC Sequoia Benchmarks," https://asc.llnl.gov/sequoia/benchmarks.

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

[20] 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.

[21] T. Li and L. K. John, "Run-Time Modeling and Estimation of Operating System Power Consumption," in *SIGMETRICS*, 2003, pp. 160–171.

[22] 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.

[23] NASA, "NAS Parallel Benchmarks," http://www.nas.nasa.gov/Resources/Software/npb.ht

[24] 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.

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

[26] C. Liao and B. Chapman, "A Compile-time Cost Model for OpenMP," in *Proceedings of International Parallel and Distributed Proceeding Symposium*, 2007.

[27] G. Tournavitis, Z. Wang, B. Franke, and M. F. OBoyle, "Towards a Holistic Approach to Auto-Parallelization Integrating Profile-Driven Parallelism Detection and Machine-Learning Based Mapping," in *Proceedings of Programming Language Design and Implementation*, 2009.

[28] S. W. Son, K. Malkowski, G. Chen, M. Kandemir, and P. Raghavan, "Reducing Energy Consumption of Parallel Sparse Matrix Applications Through Integrated Link/CPU Voltage Scaling," *Journal of Supercomputing*, 2007.

Dong Li received the Ph.D. degree in Computer Science from Virginia Tech in 2010. He is currently a research staff member in Oak Ridge National Laboratory. His research interests include power-aware computing, performance modeling and optimization, computer architecture and memory systems. He is a member of the IEEE and the ACM.

Bronis R. de Supinski is the principal investigator and leader of the Exascale Computing Technologies (ExaCT) project and the co-leader of the Advanced Simulation and Computing (ASC) program's Application Development Environment and Performance Team (ADEPT) at Lawrence Livermore National Laboratory (LLNL). He is also an Adjunct Associate Professor in the Department of Computer Science and Engineering at Texas A&M University. His research interests include high performance computer architectures, performance modeling and analysis, message passing implementations and programming models. Throughout his career, Bronis has won several awards, including the prestigious Gordon Bell Prize in 2005 and 2006. He is a member of the ACM and the IEEE Computer Society.

Martin Schulz is a Computer Scientist at the Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory (LLNL). His research interests include parallel and distributed architectures and applications; performance monitoring, modeling and analysis; memory system optimization; parallel programming paradigms; tool support for parallel programming; power efficiency for parallel systems; and fault tolerance at the application and system level. In his position at LLNL he especially focuses on the issue of scalability for parallel applications, code correctness tools, and parallel performance analyzer as well as scalable tool infrastructures to support these efforts.

Dimitrios S. Nikolopoulos is a Professor of Electronics, Electrical Engineering and Computer Science at Queen's University of Belfast, where he holds the Chair in High Performance and Distributed Computing (HPDC) and serves as Director of Research in the HPDC area. Prof. Nikolopoulos is also an Affiliated Faculty Member of FORTH–ICS, in Greece. His research interests span all aspects of parallel system software and the hardware-software interface of parallel computer architectures. He has received numerous awards recognizing his research contributions, including NSF and DOE Faculty Career Awards, an IBM Faculty Award, a Marie Curie Fellowship, a Fellowship from the High Performance and Embedded Architectures and Compilation (HiPEAC) European Network of Excellence, and six conference best paper awards, including those of SC, PPOPP and IPDPS. Prof. Nikolopoulos is a Senior Member of IEEE and a Senior Member of ACM. He received the Ph.D. in Computer Engineering in 2000 from the University of Patras.

Kirk W. Cameron received the Ph.D. in Computer Science from Louisiana State University (2000) and B.S. in Mathematics from the University of Florida (1994). The central theme of his research is to improve performance and power efficiency in high performance computing (HPC) systems and applications. Since 2005, he has been an Associate Professor at Virginia Tech and was named a Faculty Research Fellow for the VT College of Engineering in 2007. Other accolades for his research include NSF and DOE Career Awards and an IBM Faculty Award. Prof. Cameron is a pioneer and leading expert in Green Computing. His advanced power measurement software infrastructure (PowerPack) is used by dozens of research groups around the world. His power management software, Granola, is used by hundreds of thousands of people in more than 160 countries. Cameron is also a Green IT columnist for IEEE Computer, Green500 co-founder, founding member of SPECPower, EPA consultant, Uptime Institute Fellow, and co-founder of software power management startup company MiserWare.