CPU MISER: A performance-directed, run-time system for power-aware clusters

Conference Paper · October 2007

DOI: 10.1109/ICPP.2007.29 · Source: IEEE Xplore

CITATIONS
173

READS
113

4 authors, including:

Rong none Ge
Clemson University
52 PUBLICATIONS **1,637** CITATIONS
SEE PROFILE

Xizhou Feng
Clemson University
36 PUBLICATIONS **1,843** CITATIONS
SEE PROFILE

Wu Feng
Virginia Polytechnic Institute and State University
394 PUBLICATIONS **6,644** CITATIONS
SEE PROFILE

CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters

Rong Ge, Xizhou Feng, Wuchun Feng, and Kirk W. Cameron
Center for High-End Computing Systems
Department of Computer Science
Virginia Tech, Blacksburg, VA 24061
{ge,fengx,feng,cameron}@cs.vt.edu

Abstract

Performance and power are critical design constraints in today's high-end computing systems. Reducing power consumption without impacting system performance is a challenge for the HPC community. We present a run-time system (CPU MISER) and an integrated performance model for performance-directed, power-aware cluster computing. CPU MISER supports system-wide, application-independent, fine-grain, dynamic voltage and frequency scaling (DVFS) based power management for a generic power-aware cluster. Experimental results show that CPU MISER can achieve as much as 20% energy savings for the NAS parallel benchmarks. In addition to energy savings, CPU MISER is able to constrain performance loss for most applications within user-specified limits. These constraints are achieved through accurate performance modeling and prediction, coupled with advanced control techniques.

1 Introduction

By clustering tens of thousands of power-hungry components, today's high-end systems deliver incredible peak performance but consume tremendous amounts of electric power. For example, three of the top 10 systems in the Top500 list[1] — Blue Gene/L, ASC Purple, and NASA Columbia — consume 2.5, 7.6, and 3.4 megawatts of peak power respectively[2]. This amount of power consumption can result in operating costs that exceed acquisition costs. The heat generated can elevate ambient temperature and increase failure rates.

Reducing the power consumption of these systems is necessary, but reducing performance substantially is unacceptable. The high-performance, power-aware computing (HPPAC) approach attempts to reduce power while maintaining performance. This approach leverages power-aware components that support multiple power/performance modes and power-aware schedulers that dynamically control the time components spend in each mode. The challenge for power-aware schedulers is to place components in low power modes only when this will not reduce performance. Several research groups have shown that clever scheduling of CPU power modes using Dynamic Voltage and Frequency Scaling (DVFS) can save significant amounts of total system energy for parallel applications [9, 8, 12, 6].

Two types of DVFS schedulers have been implemented for power-aware clusters: off-line trace based scheduling [9] and run-time profiling based scheduling [12, 6]. Off-line techniques provide a good basis for comparison to evaluate the effectiveness of runtime techniques. Run-time techniques are challenging since effective scheduling requires accurate prediction of the effects of power modes on future phases of the application without any a priori information. False prediction may have dire consequences for performance or energy efficiency.

Current run-time DVFS techniques for HPC track and predict either MIPS-based metrics [11, 12] or communication phases (i.e. occurrences of MPI calls) [6]. Both techniques have been shown to reduce energy with reasonable performance loss. However, MIPS-based metrics use throughput as a performance measure which may not track the actual execution time and performance impact of DVFS on applications. On the other hand, intercepting MPI calls can predict communication phases accurately but this technique ignores other memory- or IO-bound phases that provide additional opportunities for power and energy savings.

This paper describes a new run-time scheduler, named CPU MISER (which is short for CPU Management Infra-Structure for Energy Reduction), that supports system-wide, application-independent, fine-grained, DVFS-based

[1] Source: The Top500 supercomputer sites, http://www.top500.org/.
[2] Source: The Green500 List, http://www.green500.org/.

power management for generic power-aware clusters. The contributions of CPU MISER include:

- System-level management of power consumption and performance. CPU MISER can optimize for performance and power on multi-core, multi-processor systems.
- Exploitation of several types of inefficient phases including memory accesses, IO accesses, and system idle under power and performance constraints.
- Completely automated run-time DVFS scheduling. No user intervention required.
- Integrated, accurate DVFS performance prediction model that allows users to specify acceptable performance loss for an application relative to application peak performance.

This paper is organized as follows. §2 discusses related work on DVFS-based power-aware computing. In §3, we present the theoretical foundations of CPU MISER, including the underlying performance model, workload prediction, and performance control. §4 describes the system implementation of CPU MISER. Experimental results on a power-aware cluster are presented and analyzed in §5. Finally, we summarize our findings and conclusions for CPU MISER in §6.

2 Related Work

DVFS work originated in the embedded and real time systems community [4, 5]. Later work applied similar techniques to the data center as power became a critical issue for large commercial server farms [18, 10, 1, 3]. Power-aware high-performance computing attempted to develop new techniques that save power and energy without impacting performance in parallel, non-interactive scientific applications.

Initially, off-line trace based techniques were proposed [2, 9, 13]. The basic off-line approach involves: source code instrumentation for performance profiling — execution with profiling — determination of appropriate processor frequencies for each phase — source code instrumentation for DVFS scheduling. Ge et al [2, 9] use PMPI to profile MPI communications. Hsu et al [13] use compiler instrumentation to profile and insert DVFS scheduling functions for sequential codes. Freeh et al [6] use PMPI to time MPI calls and then insert DVFS scheduling calls based on duration. Off-line approaches typically require manual intervention to determine the target frequency for inefficient phases.

Run-time DVFS scheduling techniques are automated and transparent to end users. Hsu and Feng [12] proposed the β-adaption algorithm to automatically adapt the voltage and frequency for energy savings at run-time. Lim et al [14] implemented a run-time scheduler that intercepts the MPI calls to identify communication bound phases in MPI programs. Wu et al [17] made use of a dynamic compiler to monitor the memory bound regions in sequential codes for power reduction. In addition, there existed an interval-based DVFS scheduler named CPUSPEED[3] for Linux distributions. CPUSPEED adjusts CPU power/performance modes based on the CPU utilization during the past interval.

The closest work to ours are CPUSPEED and the work by Hsu and Feng [12]. Each of these techniques exploit all possible CPU slackness including MPI communication and memory access delays. Hsu et al and CPUSPEED assume slack opportunities correlate directly to MIPS and CPU utilization. Our work differs from these other approaches in the following aspects. First, our work is based on an accurate performance model that quantifies the effects of power/performance modes on workload execution at finer granularity. Second, our work explicitly controls the performance by improving workload prediction and reducing performance loss due to false prediction.

The performance model we used in this paper is inspired by previous work [4, 17, 7] that decompose the workload into on-chip and off-chip memory accesses and make DVFS decisions based on the ratio of off-chip memory access to on-chip memory access. However, these models are not directly applicable to workloads with communication or other IO phases. On the contrary, our model integrates the effects of communication and IO phases on performance and makes DVFS decisions based on the index of CPU intensiveness, thereby including such phases as opportunities for power savings.

3 The Methodology

For a DVFS-based power aware cluster, we assume each of its compute node has N power/performance modes or processor frequencies available $\{f_1, f_2, \ldots, f_N\}$ satisfying $f_1 < f_2 < \ldots < f_N = f_{\max}$.

As we already know, by switching the processor from the highest frequency $f_{\max}$ to a lower frequency f, we can dramatically reduce the processor's power consumption. However, depending on whether the workload is CPU bound, reducing processor frequency may significantly decrease the performance as well.

Considering a generic application, we can represent its entire workload as a sequence of M execution phases over time, i.e. (w_1, t_1), (w_2, t_2), ..., (w_M, t_M), where w_i is the workload in the i^{th} phase and t_i is the time duration to compute w_i at the highest frequency $f_{\max}$.

As different workload characteristics requires different power-performance mode for optimal power-performance

[3] http://carlthompson.net/software/cpuspeed

efficiency, the goal of a system-wide DVFS scheduler is to identify each execution phase, quantify its workload characteristics, and then switch the system to the most appropriate power/performance mode.

To derive a generic methodology for designing an automatic, performance-directed, system-wide DVFS scheduler, we formulate the δ-constrained DVFS scheduling problem as follow:

Given a power-aware system and a workload W, schedule a sequence of CPU frequencies over time that is guaranteed to finish computing the workload within a time duration $(1+\delta^*)\cdot T$ and minimizes the total energy consumption.

Here δ^* is a user-specified performance loss constraint (such as 5%) and T is the execution time when the system is continuously running at its highest frequency $f_{\max}$.

Co-scheduling power and performance is a complicated problem. However, empirical observations show that CPU power decreases drastically as the CPU frequency decreases while the performance decreases at a much slower rate. This implies that as long as the performance loss is relative small, the lower frequency, the less the energy consumption. Hence, heuristically, if we schedule a minimum frequency for every execution phase that satisfies the performance constraint, we can have an approximated solution for the δ-constraint DVFS scheduling problem.

Meanwhile, because it is difficult to detect the phases boundaries at run-time, we approximate each execution phase with a series of time intervals and then schedule the power/performance modes base on the workload characteristics during each time interval.

Therefore, we decompose the task of of designing a performance-directed, system-wide DVFS scheduler into four subtasks: (1) instrumenting/characterizing the workload during each time interval; (2) estimating the time needed to compute a given workload at a specific frequency; (3) predicting the workload in the next time interval; and (4) scheduling an appropriate frequency for the next interval to minimize both energy consumption and performance loss.

To solve these subtasks, we first describe a performance model that captures the correlations between workload, frequency, and performance loss due to frequency scaling. Then, we describe techniques for workload prediction and performance control.

3.1 Performance Model

At the system level, any time duration t can conceptually be broken into two parts: t_w, the time the system is executing the workload w, and t_0, the time the system is idle. Thus we have,

$$t = t_w + t_0. \tag{1}$$

Further, as described by Choi et al [4, 7], we can dissect t_w into two parts: $t_w(f_{\text{on}})$, the cpu frequency dependent part, and $t_w(f_{\text{off}})$, the cpu frequency independent part. In short, we can express t_w as

$$t_w = t_w(f_{\text{on}}) + t_w(f_{\text{off}}). \tag{2}$$

Here, f_{on} and f_{off} refer to the on-chip and the off-chip instruction execution frequency respectively.

In Equation (2), $t_w(f_{\text{on}})$ can be estimated by $t_w(f_{\text{on}}) = w_{\text{on}} \cdot \frac{CPI_{\text{on}}}{f}$, where w_{on} is the number of on-chip memory (including register and on-chip cache) accesses, and CPI_{on} is the average cycles per on-chip access [4, 7]. $t_w(f_{\text{off}})$ can be further decomposed into main memory access time t_{mem} and IO access time t_{IO}. We approximate the main memory access time as $t_{mem} = w_{\text{mem}} \cdot \tau_{\text{mem}}$; where w_{mem} is the number of maim memory accesses, and τ_{mem} is the average memory access latency.

Thus, we can quantify the correlations between t, w, and $f_{\max}$ as

$$t = w_{\text{on}} \cdot \frac{CPI_{\text{on}}}{f_{\max}} + w_{\text{mem}} \cdot t_{\text{mem}} + t_{\text{IO}} + t_0 \tag{3}$$

Since on-chip access is often overlapped with off-chip access on modern computer architecture [17], we introduce an overlapping factor α ($0 \le \alpha \le 1$) into (3), i.e.,

$$t = \alpha \cdot w_{\text{on}} \cdot \frac{CPI_{\text{on}}}{f_{\max}} + w_{\text{mem}} \cdot t_{\text{mem}} + t_{\text{IO}} + t_0. \tag{4}$$

When the system is running at a lower frequency f, the time duration to finish the same workload w becomes:

$$t' = \alpha \cdot w_{\text{on}} \cdot \frac{CPI_{\text{on}}}{f} + w_{\text{mem}} \cdot t_{\text{mem}} + t_{\text{IO}} + t_0. \tag{5}$$

Assuming $f_{\max} \ge f$, normally $t \le t'$ and a performance loss may occur. To quantify the performance loss, we use the normalized performance loss δ, which is defined as:

$$\delta(f) = \frac{t' - t}{t}. \tag{6}$$

Substituting t and t' with (4) and (5)respectively, we obtain

$$\delta(f) = (\alpha \cdot w_{\text{on}} \cdot \frac{CPI_{\text{on}}}{f_{\max}}) \cdot \frac{1}{t} \cdot \frac{f_{\max} - f}{f} \tag{7}$$

Equation (7) indicates that performance loss is determined by both processor frequency and workload characteristics. Within the context of DVFS scheduling, we summarize the workload characteristics using κ, which is defined as

$$\kappa = (\alpha \cdot w_{\text{on}} \cdot \frac{CPI_{\text{on}}}{f_{\max}}) \cdot \frac{1}{t} \tag{8}$$

We interpret κ as an index of CPU intensiveness. When $\kappa = 1$, the workload is CPU bounded, and when $\kappa \approx 0$, the system is either idle, or memory and IO bounded.

Given a user specified performance loss bound δ^*, we identify the optimal frequency as the lowest frequency f^* that satisfies

$$f^* \geq \frac{\kappa}{\kappa + \delta^*} \cdot f_{max}. \quad (9)$$

3.2 Workload Prediction

In Equation (8) and (9), we assume the workload is given when calculating the workload characteristic index and the optimal frequency. Unfortunately, we normally do *not* know the next workload at run-time. Thus, we must predict the workload with only past information.

In this paper, we use history-based workload prediction. During each interval, we collect a set of performance events and summarize them with a single metric κ. Then we predict the κ value for the future workload using the history values of κ.

Various prediction algorithms can be used. The simplest but most commonly used technique is the PAST [16] algorithm:

$$\kappa'_{i+1} = \kappa_i, \quad (10)$$

Here κ'_{i+1} is the predicted workload at the $(i+1)^{th}$ interval and κ_i is the measured workload at the i^{th} interval. The PAST algorithm works well for slowly varying workload but incurs large performance and energy penalties for volatile workload. To better handle volatility, two kinds of enhancements have been suggested for the PAST algorithm. The first enhancement is to use the average of the history values across more intervals [15]. The second enhancement is to regress the workload either over time [4] or over the frequencies [12]. A more complicate prediction algorithm is the PID controller, which takes account of prediction error responsiveness, prediction overshooting, and workload oscillation by carefully tuning its control parameters.

In this paper, we consider an alternative algorithm called RELAX which predicts the workload using both history values and run-time profiling. The RELAX algorithm can be expressed as:

$$\kappa'_{i+1} = (1 - \lambda) \cdot \kappa'_i + \lambda \cdot \kappa_i \quad (11)$$

where κ'_i is the predicted workload at the i^{th} interval, and λ is a relaxation factor that controls how much the prediction will be relied on the current measurement κ_i.

3.3 Performance Loss Control

The DVFS scheduler overhead and false prediction of the workload are two major factors for performance loss. Given the stochastic nature of the workload, false prediction is inevitable. Consequently, dramatic performance loss occurs. For example, given a system whose highest frequency is $f_{\max} = 2.6GHz$ and lowest frequency is $f_{\min} = 1.0GHz$, considering a case where the predicted workload is $\kappa = 0$ and the actual workload is $\kappa = 1.0$, the actual performance loss during the i^{th} interval would be as high as 160%.

We address this problem by adapting the sampling interval and decreasing the weight of the intervals with possible large performance loss. Specifically, we decrease the sampling interval when the processor switches to a lower frequency, and increase the sampling interval at higher frequency. Theoretically, we set the the sampling interval $T'_s(f)$ at frequency f as:

$$T'_s(f) = max\{\frac{\delta(f)}{\delta^*} T_s,\ T_{s0}\} \quad (12)$$

Here T_s is the standard sampling interval at $f_{\max}$, δ^* is the user-specified performance loss constraint, $\delta(f)$ is the potential performance loss at frequency f, and T_{s0} is an upper bound due to practical considerations.

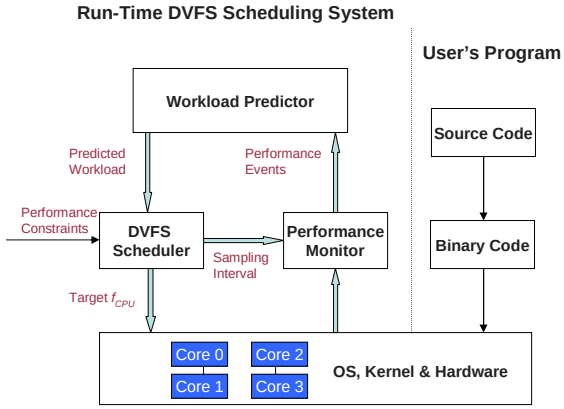


Figure 1. The implementation of CPU MISER

4 System Design

Figure 1 shows the implementation of CPU MISER, a system-wide, run-time DVFS scheduler for multicore or SMP based power aware clusters. CPU MISER consists of three components: performance monitor, workload predictor, and DVFS scheduler.

The performance monitor periodically collects performance events using hardware counters provided by modern processors during each interval. The current version of CPU MISER monitors four performance events: retired instructions, L1 data cache accesses, L2 data cache accesses, and memory data accesses[4]. The first three events capture

[4] We chose these performance events for AMD Athlon and Opteron pro-

the on-chip workload w_{on}, and the last event describes the off-chip memory access w_{mem}. Performance monitors are also used to approximate t_{IO} and t_0 from the statistics data provided by the Linux pseudo-file /proc/stat.

The workload predictor first calculates κ using the performance data collected by performance monitors and then predicts κ with a workload prediction algorithm. In CPU MISER, the memory access latency, τ_{mem} is estimated using the *lat_mem_rd* tool provided in the the LMbench microbenchmark. As it is nontrivial to estimate α and CPI_{on} separately at run-time, we approximate their product from Equation (5) and then use this product and Equation (8) to compute κ. Though CPU MISER supports several workload prediction algorithms, it uses the RELAX algorithm by default. For the RELAX algorith, CPU MISER sets the relaxation factor to an empirical value of $\lambda = 0.5$. This is semantically equivalent to the proportional mode of a **PID** controller with a proportional gain $K_{\text{P}} = 0.5$, i.e.,

$$\kappa'_{i+1} = \kappa'_i + 0.5 \cdot (\kappa_i - \kappa'_i) \tag{13}$$

The DVFS scheduler determines the target frequency for each processor based on the predicted workload κ' and modifies processor frequency using the CPUFreq interface. [5] Since the processor only supports a finite set of frequencies, we empirically normalize the calculated frequency as follows:

$\forall f^* \in [f_1, f_2]$ where f_1 and f_2 is a pair of adjacent available CPU frequencies, we set $f^* = f_2$ if $f^* \in [f_1 + \frac{f_2 - f_1}{3}, f_2]$, and $f^* = f_1$ if $f^* \in [f_1, f_1 + \frac{f_2 - f_1}{3})$.

Current multicore processors are only capable of setting the same frequency for all cores. Thus, the DVFS scheduler chooses the highest calculated frequency among all cores for the targeted processor frequency.

One additional function of the DVFS scheduler is to adapt the sample frequency based on the current frequency as described in Section 3. In our current implementation, we use two sampling intervals: when the processor is in the lowest frequency we empirically set the sampling interval to 50ms, otherwise we set the sampling interval as 250ms. We plan to study the effects of varying sampling frequency in future work.

5 Results and Discussions

5.1 Experimental Methodology

We evaluate CPU MISER on a 9-node power-aware cluster named ICE. Each ICE compute node has two dual-Core AMD Opteron Processor 2218 and 4GB main memory. Each core includes one 128KB split instruction and data L1 cache, and one 1MB L2 cache. Each processor supports 6 power/performance modes as shown in Table 1. The nodes are interconnected with Gigabit Ethernet. We run SUSE Linux (kernel version 2.6.18) on each node. We use CPUFreq for the DVFS control interface and PERFCTR for hardware counter access interface.

Table 1. Power/performance modes available on a dual core dual processor cluster ICE

Frequency (MHz)	Voltage (V)
1000	1.10
1800	1.15
2000	1.15
2200	1.20
2400	1.25
2600	1.30

The programs we have evaluated include the NAS Parallel Benchmark suite. We use MPI (Message Passing Interface) as the model of parallel computing. The MPI implementation is MPICH Version 1.2.7. We note each experiment as XX.S.NP where XX refers to the code name, S refers to the problem size, and NP refers to the number of processes. For example, FT.C.16 means running the FT code with problem size C on 16 processes. Since we used all cores on each node during the computation, only 4 nodes are needed to provide the 16 processors.

We measure the total system power (AC power) for each node using the Watts up? PRO ES power meter. We record the power profile using an additional Linux machine. The power meter samples power every $1/4$ second and outputs the data to the Linux machine via RS232 interface.

In all results, energy and performance values are normalize to the highest CPU speed (i.e. 2600MHz). In this section, we refer the energy as the total energy consumed by all the compute nodes and the performance as the elapsed wall clock time. We repeat each experiment three times and report their average values.

5.2 Experimental Results

5.2.1 Overall Energy and Performance Results

The overall energy and performance results for the NPB benchmarks are shown in Table 2. We run each code at each frequency shown in Table 1 (denoted as static DVFS control from this point), followed by one run with CPU MISER enabled, and another with CPUSpeed enabled. Table 2 shows CPU MISER can save significant energy without requiring any priori from the applications. The behavior of

cessors. For other architectures with different numbers and types of counters, the performance events monitored may require adjustment.

[5] CPUFreq Linux kernel subsystem allows users or applications to change processor frequency on the fly.

Table 2. Normalized Performance and Energy for the NAS Benchmark Suite. In each cell, the number on top is the normalized execution time, and the number on the bottom is the normalized energy. For CPU MISER, the user specified performance loss is $\delta^* = 5\%$.

Code	Frequency (MHz)						DVFS Scheduler	
	1000	1800	2000	2200	2400	2600	CPU MISER	CPUSPEED
BT.C.16	1.66	1.17	1.08	1.07	1.05	1.00	1.06	1.48
	1.06	0.88	0.84	0.90	0.96	1.00	0.95	1.07
CG.C.16	1.47	1.15	1.11	1.07	1.03	1.00	1.02	1.36
	0.98	0.88	0.88	0.91	0.94	1.00	0.93	0.95
EP.C.16	2.57	1.45	1.30	1.18	1.08	1.00	1.10	1.05
	1.57	1.07	1.00	0.98	0.98	1.00	0.99	1.01
FT.C.16	1.40	1.10	1.06	1.04	1.02	1.00	1.03	1.07
	0.92	0.84	0.83	0.88	0.94	1.00	0.89	0.92
IS.C.16	1.52	1.07	0.99	1.01	1.01	1.00	0.96	1.08
	1.01	0.82	0.79	0.85	0.93	1.00	0.80	0.78
LU.C.16	1.62	1.13	1.05	1.02	1.06	1.00	1.03	1.32
	1.03	0.86	0.83	0.86	0.96	1.00	0.94	1.01
MG.C.16	1.41	1.11	1.03	1.05	0.99	1.00	1.04	1.32
	0.92	0.84	0.81	0.87	0.90	0.98	0.92	0.92
SP.C.16	1.53	1.08	1.03	1.02	1.05	1.00	1.08	1.32
	1.00	0.84	0.81	0.87	0.96	1.00	0.98	0.97

CPU MISER is captured by the theory discussed in §3. The results also indicates that the benefits of CPU MISER vary significantly for different benchmarks. For codes with large amounts of communication and memory access, CPU MISER can save up to 20% energy with 4% performance loss. For codes that are CPU-bound (e.g. EP), CPU MISER saves little energy since reducing processor frequency would impact performance significantly.

From a graphical form of the results shown in Figure 2, we observe that CPU MISER and static DVFS control result in similar performance slowdown and energy savings for BT, CG, FT and IS while static control performs better for LU, MG, SP, and EP. However, choosing the best static processor frequency requires either a priori information about the workload or significant training and profiling. Thus, the dynamic, runtime, transparent characteristics of CPU MISER are more amenable to use in systems with changing workloads.

Here we also compare CPU MISER against CPUSPEED in Figure 2. Overall, CPU MISER saves more energy than CPUSPEED and its performance loss is controlled. In contrast, CPUSPEED may lose up to 40% performance with 7% energy increase (In our experiments, we use the default sampling interval for CPUSPEED, 1 second.) Thus, we conclude that CPUSPEED is *not* appropriate for system-wide DVFS scheduling in high-performance computing. To achieve optimal energy-performance efficiency, the rigorous theoretical analysis used in CPU MISER is necessary for scheduler design.

5.2.2 Effects of Workload Prediction Algorithms

We have implemented several workload prediction algorithms in CPU MISER. Here we compare two of them: the PAST algorithm and the RELAX algorithm. The results of these two algorithms for NPB benchmarks are shown in Figure 3. We observe that the RELAX algorithm controls performance loss better than the PAST algorithm, while the PAST algorithm may save more energy.

The PAST algorithm responds to the current workload more quickly than the RELAX algorithm, while the RELAX has a tendency to delay its decision until having observed similar workload for several intervals. These decisions dampen reactions to dramatic workload changes that last for only very short duration, thereby reducing the chances of false workload predictions. Furthermore, as described in § 3, false predictions are costly as they account for unpredictable and at times significant performance losses.

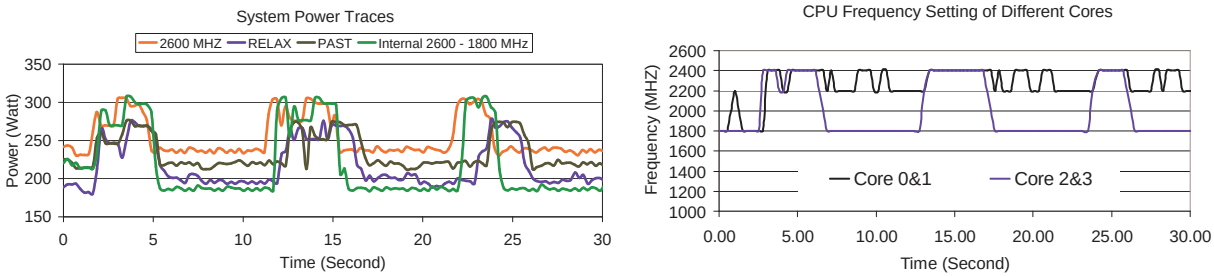
5.2.3 The Dynamic Behaviors of CPU MISER

To better understand the behavior of CPU MISER, we trace the system power consumption and CPU frequency settings on one of the compute nodes. Figure 4 shows those traces for the FT benchmark.

Figure 4-(a) shows all tested DVFS schedulers can correctly capture workload phases but different DVFS sched-

Figure 2. Performance slowdown and energy saving of CPU MISER, static control, and CPUSPEED. A negative performance slowdown indicates performance improvement and a negative energy saving indicates energy increases.

Figure 3. Performance slowdown and energy saving of CPU MISER when using the PAST algorithm and the RELAX algorithm with $\lambda = 0.5$.



(a) Traces of CPU power consumptions

(b) Traces of CPU frequencies at different cores

Figure 4. System power (a) and frequency (b) traces under CPU MISER, where (b) is with RELAX algorithm.

ulers may result in different system power consumptions. In contrast, CPU MISER not only scales down the processors during the communication phases, but also runs at a relatively lower frequency during the computation phases.

The detailed examination of CPU MISER in Figure 4-(b) shows that CPU MISER schedules CPU core 2 and core 3 to a lower frequency than core 0 and core 1. We believe that the major reason for this difference comes from the fact that core 0 is taking care of the system activities. Meanwhile, since two cores on the same processor have to be run at the same frequency, there are certain amount of power inefficiencies on core 1 due to co-scheduling with core 0, though CPU MISER does correctly predict the best frequencies for core 0 and core 1.

6 Conclusions and Future Work

In summary, this paper presents the methodology, design, and evaluation of performance-directed, system-wide, run-time DVFS schedulers for high performance computing. We have evaluated CPU MISER, a run-time DVFS scheduler designed with the proposed methodology, on a real power-aware cluster.

Our experimental results show that NPB benchmarks

save up to 20% energy when using CPU MISER as the DVFS scheduler and that performance loss for most applications is within the user defined limit. This implies the methodology we presented in this paper is promising for large scale deployment. We attribute these results to the underlying performance model and performance loss analysis. However, we also noticed that further tuning for CPU MISER is possible.

Given the fact that CPU MISER is built upon a generic framework, and transparent to both users and applications, we expect that it can be extended to many power-aware clusters for energy savings. In the future, we will refine runtime parameter derivation and improve the prediction accuracy. We will also further investigate the impacts of CPU MISER to more architectures and applications as well.

Acknowledgement

The authors would like to thank the National Science Foundation and the Department of Energy for sponsoring this work under grants NSF CCF-#0347683 and DOE DE-FG02-04ER25608 respectively.

References

[1] R. Bianchini and R. Rajamony. Power and energy management for server systems. *IEEE Computer*, 37(11):68–76, 2004.

[2] Kirk W. Cameron, Rong Ge, and Xizhou Feng. High-performance, power-aware distributed computing for scientific applications. *IEEE Computer*, 38(11):40–47, 2005.

[3] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *the 17th International Conference on Supercomputing*, 2003.

[4] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, 2004.

[5] Keith I. Farkas, Jason Flinn, Godmar Back, Dirk Grunwald, and Jennifer M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proceedings of the 2000 ACM SIGMETRICS (SIGMETRICS'00)*, 2000.

[6] Vincent W. Freeh and David K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'05)*, 2005.

[7] Rong Ge and Kirk W. Cameron. Power-aware speedup. In *The 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*, Long Beach, CA, 2007.

[8] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Improvement of power-performance efficiency for high-end computing. In *The 1st Workshop on High-Performance, Power-Aware Computing*, 2005.

[9] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Proceedings of the ACM/IEEE Supercomputing 2005 (SC'05)*, 2005.

[10] Jerry Hom and Ulrich Kremer. Inter-program optimizations for conserving disk energy. In *Proceedings of the 2005 international symposium on Low power electronics and design (ISLPED'05)*, 2005.

[11] Chung-Hsing Hsu and Wu chun Feng. Effective dynamic voltage scaling through cpu-boundedness detection. In *The 4th international workshop on Power-aware computer systems (PACS'04)*, 2004.

[12] Chung-Hsing Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the ACM/IEEE Supercomputing 2005 (SC'05)*, 2005.

[13] Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation (PLDI'03)*, 2003.

[14] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal. Mpi and communication - adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *Proceedings of the ACM/IEEE Supercomputing 2006 (SC'06)*, 2006.

[15] Ankush Varma, Brinda Ganesh, Mainak Sen, Suchismita Roy Choudhury, Lakshmi Srinivasan, and Jacob Bruce. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems (CASE'03)*, 2003.

[16] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating System Design and Implementation (OSDI'94)*, November 1994.

[17] Qiang Wu, Margaret Martonosi, Douglas W. Clark, Vijay Janapa Reddi, Dan Connors, Youfeng Wu, Jin Lee, and David Brooks. Dynamic-compiler-driven control for microprocessor energy and performance. *IEEE Micro*, 26(1):119–129, 2006.

[18] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberley Keeton, and John Wilkes. Hibernator: Helping disk array sleep through the winter. In *the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, 2005.