Coordinated Power Management of Periodic Real-Time Tasks on Chip Multiprocessors

Vinay Devadas Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
{vdevadas, aydin}@cs.gmu.edu

***Abstract*—In this paper, we undertake the problem of minimizing *system-level* energy on chip-multicore processors (CMPs) executing a periodic real-time workload. Our framework has two components: i.) a static phase that selects a subset of cores upon which the workload can be executed without dissipating excessive *static* power and performs task-to-core allocation, ii.) a dynamic phase that involves managing the selected cores at run-time through coordinated power management framework that exploits Dynamic Voltage and Frequency Scaling (DVFS) as well as multiple idle states offered by modern CMP architectures, to reduce the *dynamic* power. We explicitly consider the unique traits of the currently available CMP architectures that distinguish them from multiprocessors, including the unique voltage level shared by the cores and its implications for DVFS. We identify the *global energy-efficient frequency* which indicates the minimum frequency level at which concurrent execution on multiple cores should take place to preserve the efficiency of DVFS. Then we propose two algorithms CVFS and CVFS* to minimize the dynamic energy consumption through concerted use of DVFS and idle states. Our experimental evaluation indicates that our framework can provide significant gains in system energy.**

***Keywords*-Real-time Scheduling, Chip Multiprocessors, Energy Management**

I. Introduction

The chip multiprocessors (CMPs) that offer multiple processing cores on a single chip have quickly become prevalent in the computing landscape. Major chip makers (Intel, AMD, Sun) have now several CMP lines with 2, 4 or 8 cores [9], [15], [17], [19], [24]. Further, extensive research activity is underway to build chips with potentially hundreds of cores (or, *many-core* systems [7], [20]). This development has important implications for real-time embedded applications that will execute on these high performance architectures. Energy management has been a very active research area in the recent past and one of the main motivating factors leading to CMP architectures was the unsustainable ever-increasing frequency and power density trends of traditional single-core architectures [7]. As a result, the CMPs come equipped with a variety of advanced power management features (e.g. Dynamic Voltage and Frequency Scaling (DVFS) and multiple idle states (e.g. *Halt, Sleep, Off*)) [28], and most comply with the ACPI standard [34] endorsed by the industry.

As part of the recent energy management research, several papers proposed DVFS-based solutions for real-time embedded systems running on conventional multi-processor platforms (where each processor is located on a separate chip) [1], [4], [8], [30]. These studies identify two main dimensions of the problem as *task-to-CPU allocation* and *run-time voltage scaling on individual CPUs* [1], [30]. Yet, the emerging CMP platforms have a number of unique traits which make the problem different from the multi-processor platforms. For example, while it is natural to have different voltage levels per CPU (*per-CPU DVFS* capability) in a *multi-processor* system, the tight coupling of cores on a single chip (CMP) implies that the *per-core DVFS* feature would come with severe additional circuit complexity, stability, and power delivery problems [7], [11], [20]. In fact, in the state-of-the-art commercial CMP lines the processing cores share a common voltage level. A recent study [11], based on detailed VLSI circuit simulations, suggests that the potential energy gains of *per-core* DVFS are likely to remain too modest for justifying the complicated design problems. For the next-generation *many-core* systems, it is likely that only a small number of clusters/blocks each with several cores and independent voltage regulators will be feasible [7]. Independent and effective management of such clusters (or, the so-called *voltage islands*) would be the ultimate objective in these next-generation systems [7], [11], [29].

The focus of this paper is the effective system-level energy management of a set of processing cores that share the same supply voltage and frequency (voltage island). Many state-of-art multi-core processor lines including Intel's Itanium 2, i5, i7 and Core Duo, and IBM's Power 6 and Power 7 series, have this feature [13], [14], [16], [22], [23]. While with the *Opteron* architecture AMD has started to offer an option where the frequency (but not the voltage) of the individual cores can be scaled independently [9], the systems with global voltage/frequency are likely to remain the very common in foreseeable future.

The voltage island model poses a number of challenges that only very recently started to attract attention [18],

[27], [31]. To start with, under the global voltage/frequency constraint, the core with the maximum load at a specific time becomes the main deciding factor in the overall CMP energy consumption [27], [31]. This suggests the importance of load balancing [27], [31]. However, the frequency-independent power characteristics (which limit the efficacy of DVFS) may vary over time, complicating the problem. In addition, while parallelism in general helps to save energy [4], [8], [32], the increasing core-level static power trends effectively limits the energy-efficient parallelism level: for light workloads, it can be more energy-efficient to use only a subset of cores (and put others to *Off* state), as opposed to keeping all cores in *Active* state [7], [20], [28].

Contributions. This research effort investigates various aspects of energy management in a voltage island, for a periodic real-time workload that is partitioned to processing cores, by taking into account both *static* and *dynamic* power. Our solution has two main components. The first problem in our settings is to statically select an optimal number of processing cores for the execution of the workload, to balance static and dynamic energy consumptions. Further, tasks must be allocated to the selected cores. The un-selected cores are put to *Off* states with negligible power dissipation, avoiding excessive static power that could result from using *all* cores at light workload conditions. While the problem is NP-Hard, we propose and evaluate several effective schemes with varying levels of relative complexity and performance. Our experimental evaluation indicates that by limiting the number of cores to execute the workload, substantial energy gains are possible, especially when the load drops below 25% of the total computing capacity.

We also address various aspects of run-time management of the selected cores to reduce their *dynamic* energy consumption, under global voltage assumption. First, we identify the time-dependent *global energy-efficient frequency* concept, characterizing the boundaries of effective DVFS in a voltage-island. We show how this global frequency can be re-computed at scheduling points, by taking into account the *active* cores and characteristics of tasks that will run in parallel upon them during the next interval. We also show how an idle core can be put to *Sleep* state to temporarily eliminate the dynamic power, without violating the timing constraints or incurring excessive state transition overhead.

We propose the *Coordinated Voltage and Frequency Scaling (CVFS)* algorithm to determine the feasible frequency while satisfying the energy-efficient execution across all cores. We also present an enhanced version CVFS* that adapts to the actual workload conditions at run-time. CVFS* considers not only early completions, but also exploits the implications of operating individual cores at a global frequency level typically higher than what the feasibility of the local workload requires. We experimentally investigate the performance of CVFS under different load and maximum utilization factors. We also evaluate the performance of CVFS* and report that while the gains are marginal at low-load conditions, its capability to adapt to dynamic workload variability enables up to 40% additional savings with respect to CVFS.

II. System Model

A. Task and Processor Model

We consider a set of n periodic real-time tasks $\psi = \{\tau_1 \dots \tau_n\}$ that are partitioned upon m homogeneous processing cores $C_1 \dots C_m$. We use ψ_i to denote the subset of tasks allocated to core C_i.

Each periodic task τ_i is characterized by a worst-case workload of wcc_i cycles and a period of P_i, assumed to be equal to the relative deadline of its jobs. We assume the Global DVS feature as in [18], [27], [31]: the voltage can be adjusted for all active cores uniformly, along with the frequency (up to an upper bound f_{max}). The worst-case execution time of task τ_i under frequency f, is given by $\frac{wcc_i}{f}$. We use the symbol W_i to denote the worst-case execution time of task τ_i under maximum frequency; that is, $W_i = \frac{wcc_i}{f_{max}}$.

The *base utilization* of task τ_i (under maximum frequency) is $U_i = \frac{W_i}{P_i} \leq 1.0$. Hence, the total utilization of the task set ψ is given by $U_{tot} = \sum_{i=1}^{n} U_i \leq m$. Finally, the load on core C_i is given by the total utilization of tasks allocated to C_i, namely, $\sigma_i = \sum_{\tau_j \in \psi_i} U_j \leq 1$. On each core, the preemptive *Earliest Deadline First (EDF)* scheduling policy is adopted.

B. Power Model

Advanced Configuration and Power Interface (ACPI) [34] is a unified and open power management standard introduced and endorsed by major hardware and software manufacturers such as Intel, Microsoft, HP and Toshiba. ACPI defines an *active* state in which the core executes instructions. The exact power profile in *active* state (defined as state C_0 in ACPI) will consist of *static* and *dynamic* power figures. In the *active* state, by using the power model from [2], [29], [33], we model the power consumption of a core C_i executing task τ_j as:

$$P_i(t) = P_{static} + a_j V^2 f + P_{ind}^j$$

where $a_j V^2 f$ and P_{ind}^j represent the *frequency-dependent* and *frequency-independent* components of active power, respectively. V denotes the supply voltage and f denotes the CPU clock frequency. a_j is the effective switching capacitance of τ_j. Note that the values of a_j and P_{ind}^j depend on the characteristics of the task τ_j executing on core C_i at a given time [2]. P_{static} represents the static power.

In Global DVS settings, all *active* cores are inherently constrained to operate at the same supply voltage and frequency level [18], [27], [31]. Given the almost linear relationship between supply voltage and frequency, the power consumption of the *active* core C_i at time t is given as:

$$P_i(t) = P_{static} + a_j f^3 + P_{ind}^j \tag{1}$$

The *aggregate* power consumption of all the cores varies with time and is a function of individual core states and the global operating frequency of all *active* cores. Let H be the hyperperiod of the task set ψ. The energy consumption of the voltage island over the interval $[0, H]$ is given as:

$$E = \int_0^H \sum_{i=1}^{m} P_i(t)\, dt$$

When a core is not executing any instructions, it may be put in one of the various *idle* states [34]. Each idle state has a different power consumption characteristic; as a general rule, the lower power consumption in a given idle state, the higher the time and energy overheads involved *in returning to the active state*. While the exact number of idle states varies from architecture, in this work, we assume the existence of at least the following three fundamental states that are supported by most modern multicore systems:

- *Halt* state: In this state, the execution of instructions is halted and the core clocks are gated, resulting in significant reduction in dynamic power. The core can return to *active* state almost instantaneously ($\approx 10ns$) [19], [34]. We model the power consumption on core C_i in the *halt* state as $P_i = P_{static} + P_0$, where P_0 is the reduced dynamic power.
- *Sleep* state: Here, further, the Phase Locked Loops (PLLs) are gated and $L1$ cache contents are invalidated. In this state, the dynamic power is practically eliminated thus making P_{static} the only component of power consumption. However, this saving in power consumption comes at the cost of addition overheads compared to the *halt* state. Returning to *active* state may require a few hundred microseconds *and* involves non-trivial energy overheads [24], [34].
- *Off* state: Here, the core voltage is reduced to very low levels, to make even the static power consumption negligible. CPU context is not preserved and returning to *active* state involves significant time and energy overheads [34]. Intel's new *i7* architecture achieves this very low energy consumption through *power gating* feature [19].

III. Global Energy-Efficient Frequency

Existing DVS studies for *uni-processor* systems established that the frequency-independent dynamic power (P_{ind}) implies the existence of a *energy-efficient frequency* (also called *critical frequency*) threshold below which DVS is no longer effective from the system-level energy point of view [2], [12], [33]. This is because, with decreasing frequency, the gains in frequency-dependent dynamic energy can be offset by the excessive increase in frequency-independent dynamic energy after some point. Further, as different tasks may have different power characteristics, the *energy-efficient frequency* is task-dependent [2]. In [33], the value of this *energy-efficient frequency* for task τ_i with effective switching capacitance a_i and frequency-independent power P_{ind}^i was given as $\sqrt[3]{\frac{P_{ind}^i}{2 \cdot a_i}}$.

However, in CMP platforms, with the unique voltage and frequency constraint, this concept needs to be re-visited. Consider $k \leq m$ *active* cores, where core C_i executes task τ_i. Let the set of tasks $\tau_1, \ldots, \tau_k$ run in parallel from t_1 to t_2 as shown in Figure 1.

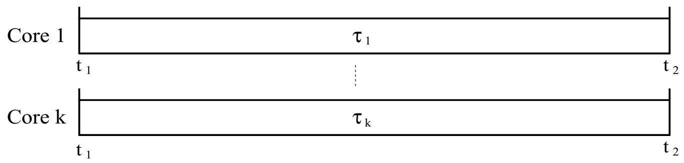


Figure 1. $\tau_1, \ldots, \tau_k$ running in parallel

During this concurrent execution, τ_i completes c_i cycles of its workload. If f denotes the global operating frequency in interval $[t_1, t_2]$, the total dynamic energy consumption in this interval is given by:

$$E' = \sum_{i=1}^{k} (a_i f^3 + P_{ind}^i) \cdot \frac{c_i}{f} = \sum_{i=1}^{k} (a_i f^2 + \frac{P_{ind}^i}{f}) \cdot c_i$$

It can be easily verified that E' is a strictly convex function of f. Thus, by setting the first derivative of E' to zero we obtain the *global energy-efficient frequency* threshold for the k *active* cores at time t as:

$$f_{ee}(t) = \sqrt[3]{\frac{P_{ind}(t)}{2 \cdot a(t)}} \tag{2}$$

where $P_{ind}(t) = \sum_{i=1}^{k} P_{ind}^i$ and $a(t) = \sum_{i=1}^{k} a_i$.

Observe that $f_{ee}(t)$ is independent of the workloads (c_i values) of tasks running in parallel. Moreover, the global energy-efficient frequency level is potentially different from the energy-efficient frequency levels of tasks executing in parallel. In other words, global energy management may mandate the use of frequency levels that are *below* individual tasks' energy-efficient frequency thresholds.

Remark 1: The global energy-efficient frequency threshold depends on the frequency-dependent active power and effective switching capacitance of the *set* of currently executing tasks on all *active* cores. Since the set of tasks executing

in parallel changes with time, the global energy-efficient frequency threshold is time-dependent.
Consequently, at the scheduling points that correspond to job completion, dispatch and preemption events, the global energy-efficient frequency should be re-computed. This operation will take at most $O(m)$ time at each scheduling point.

Remark 2: The timing constraints of the task set may require using a frequency-level higher than $f_{ee}(t)$ at time t. The time-dependent global energy-efficient frequency level indicates a lower bound that should not be violated even if timing constraints allow.

IV. Components of Coordinated Power Management

Effective and coordinated power management of multiple processing cores to execute a given workload involves two main dimensions: statically making core activation and task-to-core allocation decisions, and dynamically managing the activated cores. Note that, since we assume a partitioned-based approach, the allocation of the periodic tasks to cores is done statically and run-time migration of tasks is not considered.

A. Energy-efficient Core Activation and Task Allocation

In general, the number of available processing cores (m) may be greater than the minimum number of cores upon which the given real-time workload can be scheduled in feasible manner. While the early studies that exclusively focused on dynamic power [1], [4] suggested using *all* processing elements in parallel whenever possible, ever-increasing static power figures [7], [20] renders such an approach infeasible.

The power consumption of a given core can be minimized (in fact, effectively eliminated through techniques such as *power gating* in Intel i7 architecture [19]) when it is put to *off* state (Section II). In *active, halt* and *sleep* states, the static power would be consumed continuously. This is because the periodic nature of the real-time application and significant time/energy overheads associated with transitions to/from *off* state make dynamically putting a core to *off* state at run-time an unrealistic option. As a result, instead of activating a core with light workload (with corresponding static energy consumption), it would be preferable to move that workload to other cores when possible. Obviously, a correlated and major issue is to perform *task allocation* on the selected cores to preserve feasibility and prepare favorable initial conditions for run-time management of dynamic energy.

Thus, the offline phase can be seen as an integrated component that decides on task-to-core allocations while keeping an eye on total (i.e. static+dynamic) potential energy consumption. The $k \leq m$ cores selected by this phase will be activated and then will be managed by the run-time component. The remaining $(m-k)$ cores are put to *off* state with negligible power consumption.

B. Run-time Power Management of Active Cores

The run-time management of the selected $k \leq m$ cores involves the use of Global Voltage Scaling as well as selectively putting some cores to *halt* and *sleep* states (Section II) to reduce dynamic energy. To start with, the global frequency level that determines the dynamic power consumption at time t is decided by the highest performance level required by any core in *active* state at time t (Equation (1)). This requires both closely monitoring the workload conditions on all cores and exploiting the available idle states whenever possible. As an example, if the core that requires highest performance level (to guarantee the feasibility of its worklad) is put to *halt* or *sleep* state temporarily, the frequency can be reduced to the next highest performance level required by any of the remaining active cores during that interval. In addition, putting any core to *halt* and in particular *sleep* states have the potential of reducing dynamic energy consumption for all the cores through reducing the global energy-efficient frequency (Section III).

Now, we proceed with a detailed discussion of these two fundamental dimensions in Sections V and VI. Since our solution to the problem of energy-efficient core activation depends on some important dynamic energy consumption approximation formulas that are driven by the results of Section V, we first present that component.

V. Run-time Coordinated Power Management

In this section, we assume that $k \leq m$ cores are selected for the execution of the periodic workload and that task-to-core allocations are already performed by the static phase.

A. Exploiting Core Idle States at Run-time

In general, any of the k cores can be occasionally put to *halt* and *sleep* states when they have no ready task to execute, with corresponding gains in dynamic energy on the related core. While transitioning to *sleep* state provides higher dynamic energy savings, more significant time and energy overheads associated with that transition requires a more careful evaluation (Section II). In fact, there exists a minimum length of idle interval, denoted by $\mathcal{I}_{thres}$, that justifies transitioning a core to *sleep* state [18]. Thus, an idle core can be put to *sleep* state in energy-efficient manner *if and only if* its predicted length of idle interval is no less than $\mathcal{I}_{thres}$.

To preserve the feasibility of the workload, we provide a simple scheme to compute the predicted length of the idle interval. Since our framework is based on preemptive EDF which is a *non-idling* scheduling algorithm by definition, the earliest time in future an idle core will have to execute a task is constrained by the earliest next release time among all jobs allocated to that core. Note that this value can be easily computed given the periodicity of the real-time tasks. This value provides a safe lower bound on the *minimum* length of idle interval and hence can be used for making safe core state transitioning decisions. Let nrt_j denote the earliest time in future a job of task τ_j may be released. Then, at time t, the *minimum* length of idle interval for an idle core C_i is given as:

$$\delta_i(t) = min(nrt_j) - t, \quad j|\tau_j \in \psi_i$$

where ψ_i denotes the set of tasks allocated to C_i. An idle core C_i will be transitioned to *sleep* state *if and only if* $\delta_i(t) \geq \mathcal{I}_{thres}$. Following this, a timer is set to appropriately start transitioning C_i such that it will be *active* and ready to execute jobs at time $t + \delta_i(t)$ which marks the end of its idle interval. On the other hand, if $\delta_i(t) < \mathcal{I}_{thres}$ then C_i is simply put to *halt* state, which involves negligible transition overheads [34]. Finally, note that the run-time overhead of making this decision is constrained by the complexity of computing $\delta_i(t)$. On each core, one can always update the information about the next earliest job release time in the future ($min(nrt_j)$) at job release times in $O(1)$ time. Thus, core state transition decisions can be done in constant-time.

Remark 3: Core transitions to *halt* or *sleep* states not only help reduce power at the core-level but may potentially provide additional savings for the entire voltage island, since the global energy-efficient frequency may be effectively reduced.

B. Coordinated Voltage and Frequency Scaling (CVFS) Algorithm

Recall from Section III that running all the *active* cores at $f_{ee}(t)$ at all times minimizes the dynamic energy. However, obviously, this does not necessarily guarantee the feasibility of the workload. Since $f_{ee}(t)$ is time-dependent, computing the *optimal* feasible frequency $f(t) \geq f_{ee}(t)$ to minimize energy in the long-run poses great challenges. Hence, we take a more direct but efficient approach.

The feasibility on each *active* core C_i is guaranteed as long as its operational frequency is no smaller than is total load (utilization) [3], [25]. In other words, ensuring that $f(t) \geq \sigma_i$ at all times preserves the feasibility on core C_i. Let $\sigma(t)$ denote the largest load value among all *active* cores, i.e.,

$$\sigma(t) = max(\sigma_i), \quad i|C_i \ is \ active$$

CVFS consists in setting $f(t) = max(\sigma(t), f_{ee}(t))$ to preserve the feasibility of all *active* cores without violating the energy-efficient frequency constraint. Recall that the scheduling points and core state transitions that can potentially change the set of simultaneously executing tasks may have an impact on the global energy-efficient frequency threshold $f_{ee}(t)$. Thus, $f(t)$ needs to be re-computed at these important events. Note that the new value of $f(t)$ can be evaluated in time $O(m)$ at each scheduling point.

C. CVFS: Adapting to Dynamic Load Conditions*

CVFS is based on using the *static* load values of active cores at run-time. The load $\sigma_i = \frac{W_i}{P_i}$ corresponds to the worst-case utilization of the task set ψ_i on the core C_i. While this is a safe approach, there are potential benefits in computing the *instantaneous* load σ_i^*, which may differ from σ_i for two reasons:

- Some jobs may not take their worst-case cycles and complete early. Due to this unused CPU time, in some intervals, the instantaneous load may be less than σ_i.
- Due to the constraints imposed by $f_{ee}(t)$ and global voltage/frequency, a given core may be forced to execute at frequency levels higher than what is necessary to preserve its own feasibility. Hence, its remaining workload may be lower than σ_i in some intervals.

The algorithm CVFS* is based on maintaining a reasonably accurate estimation of the core-level instantaneous loads σ_i^* and reducing the frequency below what is suggested by CVFS when the conditions allow.

Exploiting task early completions. In this direction, we extend the well-known *cycle-conserving EDF (cc-EDF)* algorithm (which is originally proposed for uni-processor systems [25]) to multicore environments with global energy-efficient frequency awareness. Specifically, for each task τ_j on core C_i we define $u_j(t)$ as its *effective load* at time t. The rules to update u_j on core C_i are given as [25]:

- When a job of τ_j is released, u_j is reset to $\frac{W_j}{P_j}$.
- When a job of task τ_j released at time r completes after executing $acc_j \leq wcc_j$ cycles, it has effectively consumed acc_j CPU cycles in the interval $[r, r + P_j]$. Thus, the effective load of τ_j over this interval is $\frac{acc_j}{f_{max} \cdot P_j}$. Hence, when a job of τ_j completes, u_j is set to $\frac{acc_j}{f_{max} \cdot P_j}$.

Observe that u_j is reset to $\frac{W_j}{P_j}$ at every arrival of a job of τ_j. Given this, the instantaneous effective load σ_i^* on C_i is defined as $\sigma_i^* = \sum_{\tau_j \in \psi_i} u_j$. Also, let $\sigma^*(t)$ be the maximum effective load at time t among all *active* cores, i.e.

$$\sigma^*(t) = max(\sigma_i^*), \quad i|C_i \ is \ active$$

σ_i^* is updated on C_i at events corresponding to job completions and job arrivals. Now consider the frequency assignment where at time t, all *active* cores are executed at the frequency $f(t)$ given by:

$$f(t) = max(\sigma^*(t), f_{ee}(t)) \tag{3}$$

Proposition 1: At any time t, executing core C_i at $f(t) = \sigma^*(t)$ preserves the feasibility of task set ψ_i.

Proof: The feasibility of task set ψ_i is preserved as long as the operating frequency $f(t)$ on core C_i at time

t, satisfies the constraint $f(t) \geq \sigma_i^*$. This follows from the correctness of *cc-EDF* [25]. Since $\sigma^*(t) \geq \sigma_i^*$, executing core C_i at $f(t) \geq \sigma^*(t) \geq \sigma_i^*$ preserves feasibility of task set ψ_i. ■

Corollary 1: At any time t, executing all *active* cores at $f(t) = max(\sigma^*(t), f_{ee}(t))$ preserves the overall feasibility.

Refining the load estimation. Since all *active* cores are constrained to the same global voltage/frequency, typically many cores will operate at a processing frequency higher than the level necessary to guarantee the timing constraints of their *remaining* workload. This fact can be exploited to further refine the estimate of σ_i^*, providing additional energy savings. The basic principle is given below:

On core C_i, the execution of a task τ_j at a frequency $\sigma_i' > \sigma_i$ may be seen as equivalent to executing a workload $wcc_j' < wcc_j$ at speed σ_i.

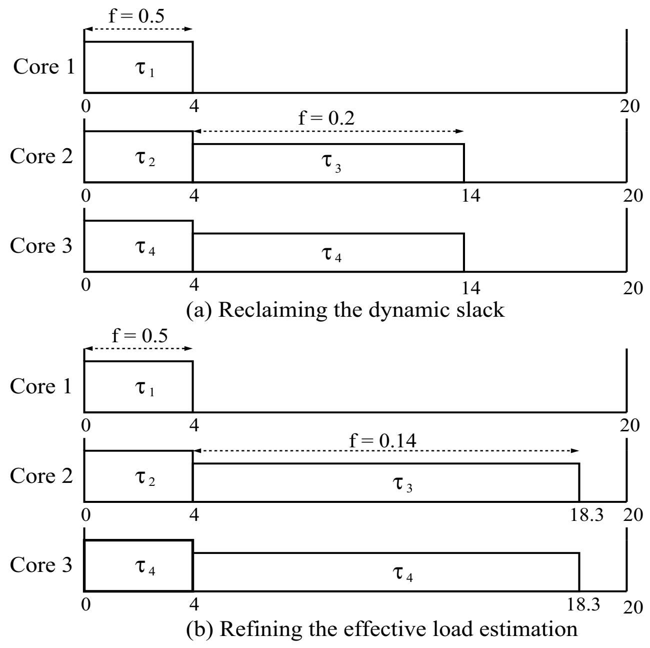


Figure 2. An example with 4 tasks and 3 cores

We give an example to illustrate this point. Figure 2 shows a CMP with three cores. C_1 has one task $\tau_1(W_1 = 10, P_1 = 20)$. C_2 has two tasks $\tau_2(W_2 = 2, P_2 = 20)$ and $\tau_3(W_3 = 2, P_3 = 20)$. C_3 has one task $\tau_4(W_4 = 4, P_4 = 40)$. Thus, the initial effective loads on the cores are given as $\sigma_1^* = 0.5$, $\sigma_2^* = 0.2$ and $\sigma_3^* = 0.1$. For simplicity, assume $a = 1$ and $P_{ind} = 0$ for all tasks. We assume $f_{max} = 1.0\ GHz$. We will concentrate on the interval $[0, 20]$. We assume that the actual workload of all tasks is the same as their worst-case workload, with the exception of τ_1 whose actual workload is 20% of the worst-case.

At time $t = 0$, τ_1, τ_2 and τ_4 are dispatched on C_1, C_2 and C_3 respectively at $f = 0.5\ GHz$. At time $t = 4$, τ_1 and τ_2 complete. Observe that at this point, the slack reclaiming rules that are previously provided would make no change to the effective load of C_2 as τ_2 took its worst-case workload. Thus, if one estimated the effective load on a core using the previous rules, then at $t = 4$, τ_3 and τ_4 would be dispatched at $f = 0.2\ GHz$ as shown in Figure 2(a).

However, observe that in the interval $[0, 4]$, τ_2 executed 2×10^9 cycles at $f = 0.5Ghz$, which is higher than $0.2\ GHz$ which is sufficient to maintain the feasibility of the workload on C_2. Thus, one can potentially see the completion of τ_2 at $t = 4$, as an early completion at $f = 0.2\ GHz$ after consuming 0.8×10^9 cycles. Hence, at $t = 4$, σ_2^* can be set to $\frac{0.8}{20} + \frac{2}{20} = 0.14$. Thus, at $t = 4$, both τ_3 and τ_4 would be dispatched at $f = 0.14\ GHz$ as shown in Figure 2(b), increasing the energy savings.

We now describe how to update σ_i^* at run-time according to these principles. Without loss of generality, assume a job of τ_j executes on core C_i in p contiguous execution intervals, denoted by $\{e_1 \ldots e_p\}$. During each contiguous execution e_k let τ_j consume at_k units of CPU time at frequency f_k. For each e_k one can compute the workload (ac_k) τ_j would have completed *at frequency σ_i in at_k time units*, by setting $ac_k = at_k \cdot \sigma_i$. The cumulative workload completed by τ_j corresponding to the contiguous execution sequence $\{e_1 \ldots e_p\}$ is given by:

$$c_j = \sum_{k=1}^{p} ac_k = \sum_{k=1}^{p} (at_k \cdot \sigma_i)$$

The operating system can keep track of and update c_j for each task τ_j appropriately at task preemption and completion points. Thus, the rules to update u_j can be re-defined (refined) as:

- When a job of τ_j is released, set $u_j = \frac{W_j}{P_j}$.
- When a job of task τ_j completes, set $u_j = \frac{c_j}{f_{max} \cdot P_j}$.

Figure 3 shows the pseudo-code for *CVFS**. The function *AdjustFrequency()* recomputes the global energy-efficient frequency threshold based on Equation (2) in Section III and the maximum effective load $\sigma^*(t)$ among all *active* cores. The new global frequency $f(t)$ is then easily calculated by taking the maximum.

An event corresponding to either job arrival or completion may change u_j which in turn may trigger changes in the effective load of C_i and hence $f(t)$. Also, as mentioned before, events corresponding to job completions, job preemptions and core state transitions have the potential to change $f_{ee}(t)$ and hence $f(t)$. Thus, at these events *AdjustFrequency()* function is called.

Since core-level power state transitioning decisions can be made in $O(1)$ time (Section V-A), the complexity of CVFS* is determined by the complexity of *AdjustFrequency()* function. Observe that the value of σ_i^* on each core C_i can be

```
At job arrival of $\tau_j$ on core $C_i$:
1  Set $u_j = \frac{W_j}{P_j}$
2  AdjustFrequency()

At transition of core $C_i$ to active state:
1  AdjustFrequency()

At job completion of $\tau_j$ on core $C_i$:
1  Set $u_j = \frac{c_j}{f_{max} \cdot P_j}$
2  if ready queue is empty
3      Set $\delta_i(t) = min(nrt_k) - t \ k|\tau_k \in \psi_i$
4      if $\delta_i(t) \geq \mathcal{I}_{thres}$
5          Transition $C_i$ to sleep state
6          Set timer to transition $C_i$ back
7      else
8          Transition $C_i$ to halt state
9  AdjustFrequency()
```

Figure 3. The pseudo-code of CVFS*

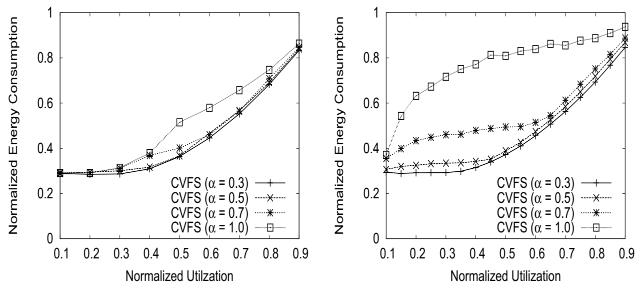
updated at job completion and job arrival events and kept track of in constant time. Thus, when *AdjustFrequency()* is called σ^* and $f_{ee}(t)$ can be re-computed in $O(m)$ time. Hence, the overall run-time complexity of CVFS* is $O(m)$ at each scheduling point.

D. Experimental Evaluation

In this section, we evaluate the performance of our algorithms through the help of a discrete-event simulator. For 2- and 8-core systems, we generated synthetic task sets each with 20 and 50 tasks, respectively. The effective switching capacitance a_i of tasks was set to 1. P^i_{ind} values were randomly chosen in the range $[0, 0.2]$. Task periods were generated randomly in the interval $[63ms, 1300ms]$ which are comparable to those seen in practice [21]. For a target total utilization value U_{tot}, we generated individual task utilizations randomly in such a way that each task utilization is no greater than a pre-defined threshold $\alpha \leq 1.0$.

Previous studies dealing with energy minimization on multi-processor systems [1], [27] showed that the maximum task utilization (denoted as α) is an important parameter for performance. As a result, we also investigated the impact of this *task utilization factor* α. In the experiments, we refer to *normalized utilization* as the quantity $\frac{U_{tot}}{m}$, where m is the number of cores on which the workload is executed. For each *normalized utilization* and α pair, we generated 1000 task sets; the data points in the plots reflect the average of these runs. The reported energy consumption values are normalized with respect to the base scheme that executes all tasks at f_{max} at all times (no power management).

First, we analyze the behavior of CVFS over the normalized utilization spectrum. In these experiments, all tasks complete their worst-case workload. Task allocation to m



(a) 2 cores (b) 8 cores

Figure 4. Impact of Utilization

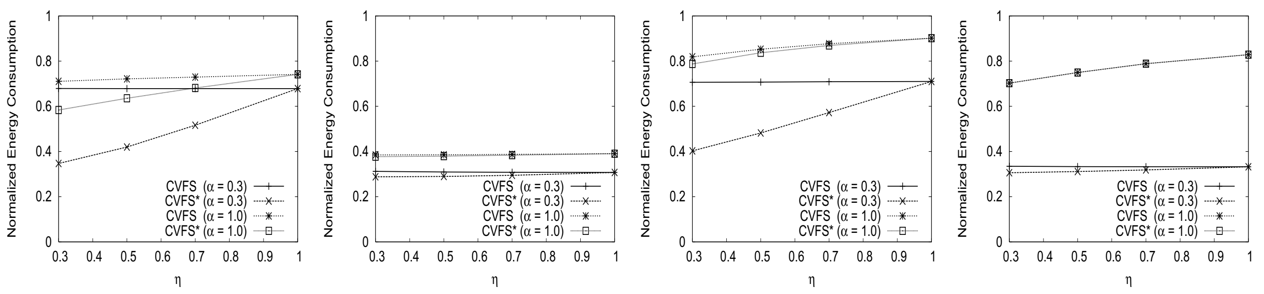
cores is done using Worst-Fit-Decreasing (WFD) heuristic which is known to generate better-balanced partitions [1], [4].

Figure 4 shows the impact of normalized system utilization on CMP with $m = 2$ cores and $m = 8$ cores for various α values. In these experiments, the actual workload of each job is equal to its worst-case. It can be seen that the CVFS scheme provides significant *overall* system energy savings. With increasing normalized utilization values, the gains of CVFS decrease as high frequency is often needed to meet the feasibility constraints. With increasing α values, the partitions created by WFD have a higher σ (maximum load among all cores) value. Since σ is one of the factors constraining $f(t)$ (Section V-B), with increasing α values the relative gains of CVFS tend to decrease.

Figure 5 shows the impact of workload variability on the schemes. We define η as the ratio of *average-case execution cycles* to *worst-case execution cycles* and use it to model the notion of dynamic workload variability. The lower the η ratio, the more the actual workload deviates from the worst case workload. For a specific value of η, the actual execution cycles are generated randomly using normal distribution.

Figures 5(a) and (c) show the impact of varying η for 2-core and 8-core systems respectively, with normalized utilization fixed to a high value (0.8). With decreasing η, the gains of CVFS* over CVFS is prominent, in particular for the case where $\alpha = 0.3$. This is due to the run-time effective load adjustments of CVFS* which provides additional DVFS opportunities and hence better energy savings compared to CVFS. For the same utilization value, higher α values tend to create more unbalanced partitions relative to lower α values. In CMP systems where all cores are constrained to operate at the same frequency, this limits the opportunities to exploit dynamic workload variability. As such, the gains of CVFS* over CVFS decreases with increasing α values.

Figures 5(b) and (d) show the impact of varying η for 2-core and 8-core systems respectively, with normalized utilization fixed to a low value (0.4). In this case, irrespective of α values, the gains provided by CVFS* over CVFS are rather small. Recall that $f(t)$ is constrained by σ and the



a. 2 cores (norm. util. 0.8) b. 2 cores (norm. util. 0.4) c. 8 cores (norm. util. 0.8) d. 8 cores (norm. util. 0.4)

Figure 5. Impact of workload variability

global energy-efficient frequency threshold $f_{ee}(t)$ (Section V-B). At low normalized utilization values, $f(t)$ is predominantly constrained by $f_{ee}(t)$, hence the run-time adaptations of CVFS* do not provide significant benefits compared to CVFS.

VI. Energy-Efficient Core Activation and Task Allocation

In this section, we elaborate on the important problem introduced in Section IV-A, namely selecting $k \leq m$ number of cores for the execution of the workload to minimize expected energy while preserving the feasibility through a proper task allocation on these k cores. Clearly, since determining feasibility of a workload on a fixed number of processors is NP-Hard in the strong sense, one cannot hope for an efficient and optimal solution to this problem.

Recall from Section V-B that at any given time the unique frequency for all *active* cores is given as $f(t) = max(\sigma, f_{ee}(t))$. Hence, starting with initial task allocations (partitions) that are reasonably balanced and adjusting these to maintain a balance between static and dynamic power consumptions is a promising approach. In fact, minimizing the maximum load among cores is also in line with existing multiprocessor and multicore energy management results [1], [4], [27]. Among task allocation heuristics, the Worst-Fit Decreasing (WFD) algorithm is known to typically yield well-balanced partitions where the maximum load on any core is small [1], [4]. Assuming that the tasks are already sorted in non-increasing order according to their utilization values, WFD allocates tasks one by one to the core with the least load at a time. For this specific problem, WFD is equivalent to the well-known *List Scheduling Algorithm (LST)* where independent tasks each with a given size in the range $[0, 1]$ are partitioned to m CPUs each with unit capacity. The result in [10] implies that the maximum load among all cores generated by LST (and equivalently WFD in our setting) is no more than $\frac{4}{3}$ times that of the optimal. As a result, the first step of our framework will consist in generating an *initial* partition on all m cores through WFD, before transforming this initial schedule into a *final* and a more energy-efficient partition with possibly a smaller number of active cores.

Having an efficient mechanism to evaluate the *expected energy* consumption of a given partition in static phase is an important component of our approach. Let $\mathcal{P}_k$ be a feasible partitioning of task set ψ to $k \leq m$ cores. Since only k cores have tasks allocated to them, the remaining $(m - k)$ cores can be put to *off* state. Thus, the static energy consumption resulting from partition $\mathcal{P}_k$ during the hyperperiod H is given as:

$$E_s(\mathcal{P}_k) = k \cdot P_{static} \cdot H$$

Since the global unique frequency at time t, $f(t)$, is time-dependent and further depends on the set of tasks executing in parallel at any given time, it is very difficult, if not impossible, to have an accurate figure for the dynamic energy consumption of the task set ψ, in advance. We estimate the dynamic energy consumption of $\mathcal{P}_k$ by calculating the weighted average value of $f(t)$ in the interval $[0, H]$. Let F_{ee} denote the weighted average of all $f_{ee}(t)$ values in the interval $[0, H]$. We approximate F_{ee} as:

$$F_{ee} = \sqrt[3]{\frac{P^*_{ind}}{2 \cdot a^*}}$$

where, $P^*_{ind} = \sum_{i=1}^{n}(U_i \cdot P^i_{ind})$ and $a^* = \sum_{i=1}^{n}(U_i \cdot a_i)$. Recall from Equation (2) that the global energy-efficient frequency at any given time is determined by the ratio of P^i_{ind} and a_i values of tasks. Hence, it is natural to expect that tasks with large utilization values will have a higher contribution to F_{ee} on the *average*. Given this, the weighted average of all $f(t)$ values in the interval $[0, H]$ can be approximated as:

$$F = max(\sigma, F_{ee})$$

The expected dynamic energy consumption of task set ψ over partition $\mathcal{P}_k$ is then calculated as:

$$E_d(\mathcal{P}_k) = \sum_{i=1}^{n}(a_i F^3 + P^i_{ind}) \cdot \frac{U_i}{F} \cdot H$$

Notice that different partitions may produce different F values and thus have different expected dynamic energy consumptions. The total expected energy consumption of $\mathcal{P}_k$ is:

$$E_{exp}(\mathcal{P}_k) = E_s(\mathcal{P}_k) + E_d(\mathcal{P}_k) \quad (4)$$

At this point, we are ready to present three schemes developed for determining the number of active cores.

Sequential-Search (SS) Algorithm. The minimum number of cores necessary to execute a workload with total utilization U_{tot} in feasible manner is $\lceil U_{tot} \rceil$. *SS* exhaustively considers every possible k in the range $[\lceil U_{tot} \rceil, m]$ and for each such k it generates a partition $\mathcal{P}_k$ using WFD. If $\mathcal{P}_k$ is a feasible partition then the algorithm computes the expected energy consumption of $\mathcal{P}_k$ using Equation (4). The k value corresponding to the partition with the least E_{exp} is returned. Figure 6 gives the pseudo-code.

```
1  for each $k$ in the range $[\lceil U_{tot} \rceil, m]$ do
2     Determine partition $\mathcal{P}_k$ using WFD
3     if $\mathcal{P}_k$ is feasible
4        Compute $E_{exp}(\mathcal{P}_k)$
5  Select the partition $\mathcal{P}_k$ yielding the minimum $E_{exp}$
```

Figure 6. Algorithm *SS*

Complexity: SS has at most m iterations. In each iteration the algorithm has to execute worst-fit decreasing (which takes $O(n \log m)$ time) and calculate E_{exp} from Equation (4) (which takes $O(n)$ time). Thus, the overall complexity of *SS* is $O(nm \log m)$.

Greedy Load Balancing (GLB) Algorithm. *GLB* invokes WFD only *once* on all m cores. Working on the resulting partitioning, *GLB* tries to free the least loaded core, by simply moving *all* tasks on the least loaded core to the second least loaded core, if and only if doing so preserves the feasibility of the workload *and* does not increase the *expected* energy consumption, computed through Equation (4). The algorithm is re-invoked iteratively for the remaining cores until such a block move of tasks is no longer possible.

In the pseudo-code given in Figure 7, $\mathcal{P}_k(\psi_i)$ represents the set of tasks allocated to core C_i in partition $\mathcal{P}_k$, whereas $\mathcal{P}_k(\sigma_i)$ denotes the load on C_i in $\mathcal{P}_k$.

Complexity: GLB invokes WFD once on all m cores (which takes $O(n \log m)$ time). Following this, *GLB* has at most m iterations (Lines 3-11) where calculating E_{exp} takes $O(n)$ time and re-arranging the position of the second least loaded core, after moving the workload from the least loaded core to it, can be done in $O(\log m)$ time. Thus, the overall complexity of *GLB* is $O(n \log m + m \log m + mn) = O(mn)$.

```
1  $\mathcal{P}_m$ = Partition obtained through WFD on $m$ cores
2  $k = min(m, n)$
3  while $(k > 1)$
4     src = index of the core with minimum load
5     des = index of the core with second minimum load
6     if $(\sigma_{src} + \sigma_{des} > 1)$ return $\mathcal{P}_k$
7     $\mathcal{P}_{k-1} = \mathcal{P}_k - (\mathcal{P}_k(\psi_{src}), \mathcal{P}_k(\sigma_{src}))$
8     Set $\mathcal{P}_{k-1}(\psi_{des}) = \mathcal{P}_{k-1}(\psi_{des}) \cup \mathcal{P}_k(\psi_{src})$
9     Set $\mathcal{P}_{k-1}(\sigma_{des}) = \mathcal{P}_{k-1}(\sigma_{des}) + \mathcal{P}_k(\sigma_{src})$
10    if $(E_{exp}(\mathcal{P}_{k-1}) \geq E_{exp}(\mathcal{P}_k))$ return $\mathcal{P}_k$
11    Set $k = k - 1$
12 return $\mathcal{P}_k$
```

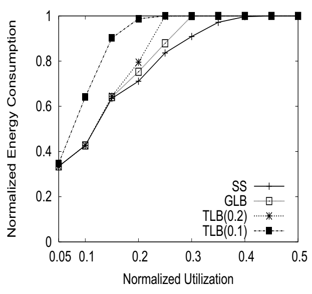
Figure 7. Algorithm *GLB*

Threshold-based Load Balancing (TLB) Algorithm. *TLB* is similar to *GLB* but does not use the expected energy formula given in Equation (4), to improve efficiency. Instead, *TLB* uses the concept of *load threshold*, wherein a partition is accepted by *TLB* as long as the minimum load on any core is no smaller than a pre-defined *threshold* value. This threshold value should be carefully chosen by the system designer to reflect an appropriate balance between static and active power consumptions. Similar to *GLB*, *TLB* first invokes WFD once on all m cores and then iteratively tries to free the least loaded core, by simply moving *all* tasks on it to the second least loaded core, if and only if the minimum load is smaller than the pre-defined *threshold and* doing so preserves the feasibility of the workload. After such a move, the algorithm is iteratively re-invoked on the new set of active cores. Figure 8 gives the pseudo-code.

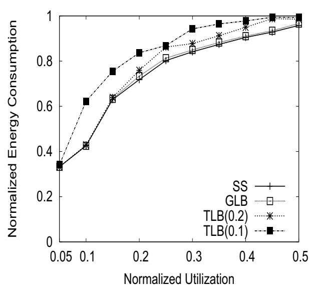
```
1  $\mathcal{P}_m$ = Partition obtained through WFD on $m$ cores
2  $k = min(m, n)$
3  while $(k > 1)$
4     src = index of the core with minimum load
5     des = index of the core with second minimum load
6     if $(\sigma_{src} > threshold$ or $\sigma_{src} + \sigma_{des} > 1)$
7        return $\mathcal{P}_k$
8     $\mathcal{P}_{k-1} = \mathcal{P}_k - (\mathcal{P}_k(\psi_{src}), \mathcal{P}_k(\sigma_{src}))$
9     Set $\mathcal{P}_{k-1}(\psi_{des}) = \mathcal{P}_{k-1}(\psi_{des}) \cup \mathcal{P}_k(\psi_{src})$
10    Set $\mathcal{P}_{k-1}(\sigma_{des}) = \mathcal{P}_{k-1}(\sigma_{des}) + \mathcal{P}_k(\sigma_{src})$
11    Set $k = k - 1$
12 return $\mathcal{P}_k$
```

Figure 8. Algorithm *TLB*

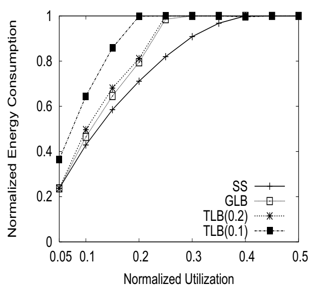
Complexity: Assuming $n \geq m$, WFD takes $O(n \log m)$ time. Following this there are at most m iterations (Lines 3-11) and in each of these iterations re-arranging the position of the second least loaded core takes $O(\log m)$ time making



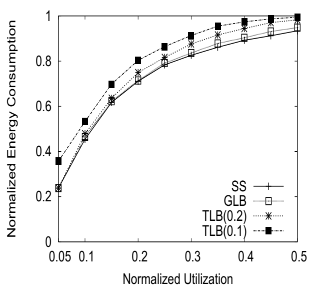
a. 4 cores ($\alpha = 0.3$)



b. 4 cores ($\alpha = 1.0$)



c. 8 cores ($\alpha = 0.3$)



d. 8 cores ($\alpha = 1.0$)

Figure 9. Comparing *SS*, *GLB* and *TLB*

the total complexity $O(n \log m)$. On the other hand, if $n < m$, WFD takes $O(n)$ time, there are only n iterations (Lines 3-11) each taking $O(\log n)$ time which gives a complexity of $O(n \log n)$. Thus, the overall complexity can be expressed as $O(n \log(min(m, n)))$.

A. Experimental Evaluation

In this section, we compare the performance of algorithms *SS*, *GLB* and *TLB*. The simulation methodology is parallel to the one described in Section V-D. We show results for 4 and 8 cores in Figure 9, the trends of which are similar. We include results for $\alpha = 0.3$ and $\alpha = 1.0$. For the *TLB* scheme, we include results with threshold values 0.1 and 0.2 that performed best in our experiments. For each core, P_{static} was set to 10% of CPU dynamic energy consumption at f_{max} [18]. In these experiments, we consider the worst-case workload for each task and once a partitioning of tasks to $k \leq m$ cores is decided by the algorithms, we execute the task set with CVFS scheme and record the energy consumption over the hyperperiod. All energy consumption values are normalized with respect to the scheme which uses all m available cores to execute the workload.

In decreasing order of performance with respect to energy savings, the algorithms can be arranged as: *SS*, *GLB*, *TLB* (with threshold 0.2) and *TLB* (with threshold 0.1). However, the better the system energy savings of a scheme, the more its computational complexity. For the *TLB* scheme, the energy benefits are sensitive to the *threshold* value. In our simulation settings we notice that a *threshold* of 0.2 provides energy benefits that are comparable to that provided by *SS*, which has high execution overhead.

At low utilization values the gains provided by the schemes are significant (easily exceeding 50%). With increasing utilization values, the dynamic energy consumption of the workload dominates static energy and hence the number of cores activated to execute the workload in energy-efficient and feasible manner approaches m. Thus, as utilization increases the benefits of schemes decrease. In fact, when the normalized utilization $\frac{U_{tot}}{m}$ exceeds 0.5, all the schemes are forced to activate all m cores to enforce feasibility or avoid excessive *dynamic power* that can result from using less number of cores at high frequencies.

It can also be seen that the benefits of the schemes decrease much quickly at lower α values compared to higher α values (Figure 9(b) and (d)). This is because, with large α values, WFD tends to generate more unbalanced initial partitions [4]. This results in more chances for finding cores with light workloads in the WFD partition; the workloads on these cores can then be transferred to the ones with high load, enabling them to switch to *off* state.

Finally, one can also see that at low normalized utilization values, the benefits of schemes are more pronounced in the case with 8 cores. This is due to the fact that with more number of cores the potential opportunities to minimize static energy by turning off cores also increase, in particular for low utilization values.

VII. Related Work

Research studies on energy management for multiprocessor real-time systems are typically based on independent DVFS capabilities of individual processors. In [4], the authors considered the problem of minimizing CPU dynamic energy with partitioned multiprocessor scheduling and EDF policy. They showed that the problem of energy-optimal partitioning is NP-Hard in the strong sense even when the total workload can fit on a single CPU. The same paper also indicated that more balanced partitions typically yield better energy savings and suggested the use of WFD partitioning scheme to balance the the load. [1] re-considered the problem for fixed-priority systems and RMS policy.

Exploiting potential and actual early task completions have been another focus point for multiprocessor systems. In [32], the authors investigate slack reclaiming strategies for global scheduling of frame-based tasks on homogeneous multiprocessors. [8] provides a 1.13-approximation algorithm for the problem of partitioning tasks to minimize the *expected energy* consumption. In [30], the authors considered the problem of energy-efficient partitioning in heterogeneous multiprocessor platforms.

Energy management of real-time tasks on CMP platforms under the global DVS constraint has started to attract the attention of the research community more recently. In [31], assuming a frame-based system where all tasks have the same deadline, the authors showed the problem is NP-Hard and provided a 2.371-approximation scheme for this simple task model. In [5], the authors proposed a power-aware scheduler for multicore systems executing a *soft* real-time workload. In [18] the authors consider a CMP system running a single real-time application modelled as a directed acyclic communication task graph (CTG). The authors effectively deploy two techniques to save energy: DVFS to reduce the dynamic energy and power shutdown of the *entire* voltage island to reduce static energy.

[27] considered the problem of energy-efficient scheduling for periodic hard real-time tasks on CMP systems. The authors proposed a scheme to re-partition tasks at run-time by resorting to task migrations, so as to create more balanced schedules that adapt to dynamic workload variability. Further, they also proposed a dynamic core scaling algorithm adjusting at run-time the number of active cores to reduce static power under the assumption that transitions between *off* and *active* states can be done instantaneously and with no additional overheads. However, in practice such transitions are rarely attractive or possible for periodic real-time applications. Moreover, the frequency-independent component of dynamic power (hence, the energy-efficient frequency) is ignored in that work. Finally, the overhead of frequent task migrations may be a concern in practice.

VIII. Conclusion

In this paper we considered the problem of system energy minimization of periodic real time tasks executing on CMP platforms with partitioning and global DVS capability. Considering a generalized power model, we derived the *global energy-efficient frequency* that depends on the set of tasks executing in parallel. We provided two schemes CVFS and CVFS* that successfully exploit global DVS and core-level idle states to increase energy savings. CVFS* has the additional capability of adapting to workload variations at run-time. We also considered the problem of determining the optimal subset of cores to execute the workload with low static power while preserving feasibility through an appropriate task allocation; and suggested three techniques (*SS, GLB, TLB*) for this purpose. Our experimental evaluation verified the effectiveness of our solutions to reduce the system energy on CMP platforms. To the best of our knowledge, this research effort is the first to consider energy-aware periodic real-time scheduling on CMP platforms, by assuming a generalized power model with takes into account frequency-dependent and -independent dynamic powers, as well as static power, while deploying safe and effective schemes based on global DVS and multiple idle state features.

Acknowledgment

This work was supported by US National Science Foundation Grants CNS-072047 and CNS-546244 (CAREER award).

References

[1] T.A. AlEnawy and H. Aydin. Energy-Aware Task Allocation for Rate Monotonic Scheduling. In *Proceedings of IEEE Real Time and Embedded Technology and Applications Symposium (RTAS),* 2005.

[2] H. Aydin, V. Devadas, and D. Zhu. System-level Energy Management for Periodic Real-Time Tasks. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS),* 2006.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware Scheduling for Periodic Real-time Tasks. In *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 584-600, May 2004.

[4] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS),* 2003.

[5] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, J. Duato. A Simple Power-Aware Scheduling for Multicore Systems when running Real-time Applications. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS),* 2008.

[6] E. Bini, G. Buttazzo and G. Lipari. Speed Modulation in Energy-Aware Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS),* 2005.

[7] S. Borkar. Thousand core chips: A Technology Perspective. In *Proceedings of the Design Automation Conference (DAC),* 2007.

[8] J-J. Chen, C-Y. Yang, H-I. Lu, and T-W. Kuo. Approximation Algorithms for Multiprocessor Energy-Efficient Scheduling of Periodic Real-Time Tasks with Uncertain Task Execution Time. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS),* 2008.

[9] J. Dorsey *et. al*. An integrated Quad-Core Opteron Processor. In *Proc. of IEEE Intl. Solid State Circuits Conference*, 2007.

[10] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. SIAM Journal on Applied Mathematics, vol. 17, no. 2. pp. 416-429, 1969.

[11] S. Herbert and D. Marculescu. Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors. In *Proc. of the Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2007.

[12] R. Jejurikar and R. Gupta. Dynamic Voltage Scaling for System-Wide Energy Minimization in Real-Time embedded systems. In *Proc. of Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2004.

[13] IBM Power 7 Overview. http://www.redbooks.ibm.com/redpapers/pdfs/redp4638.pdf

[14] Introduction to Intel Core Duo Processor Architecture. In *Intel Technology Journal*, vol 10, no. 2, 2006.

[15] Intel i7 Processor Specifications. http://www.intel.com/products/processor/corei7/specifications.htm

[16] Intel i7-800 and i5-700 Processor series. Datasheet - Volume 1. http://download.intel.com/design/processor/datashts/322164.pdf

[17] Intel Xeon Specifications. http://www.intel.com/design/intarch/xeon/specifications_xeon.htm

[18] H. Kim, H. Hong, H-S. Kim, J-H Ahn and S. Kang. Total Energy Minimization of Real-Time Tasks in an On-Chip Multiprocessor Using Dynamic Voltage Scaling Efficiency Metric. In *IEEE Transactions of Computer-Aided Design of Integrated Circuits and Systems,* vol 27, no. 11, pp. 2088-2092, 2008.

[19] R. Kumar and G. Hinton. A Family of 45nm IA Processors. In *Proc. of the Intl. Solid-State Circuits Conference*, 2009.

[20] L. Mosley. Power Delivery Challenges for Multicore Processors. In *Proc. of CARTS USA*, 2008.

[21] D.C. Locke, D. Vogel, T. Mesler. Building a Predictable Avionics Platform in Ada: a Case Study. In *Proc. of the Real-Time Systems Symposium (RTSS)*, 1991.

[22] H.-Y. McCreary, M. A. Broyles, M. S. Floyd, A. J. Geissler, S. P. Hartman, F. L. Rawson, T. J. Rosedahl, J. C. Rubio, M. S. Ware. EnergyScale for IBM POWER6 microprocessor-based systems. In *IBM Journal of Research and Development*, vol 21, no. 6, 2007.

[23] R. McGowen, C.A. Poirier, C. Bostak, J. Ignowski, M. Millican, W.H. Parks, and S. Naffziger. Power and Temperature Control on a 90-nm Itanium family processor, In *Journal of Solid-State Circuits,* 2006.

[24] A. Naveh *et al.*. Power and Thermal Management in the Intel Core Duo Processor. *Intel technology Journal*, Vol. 10, Issue 02, May 2006.

[25] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[26] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority Real-time Systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2003.

[27] E. Seo, J. Jeong, S. Park, and J. Lee. Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors. In *IEEE Trans. on Parallel and Distributed Systems,* vol. 19, no. 11 pp. 1540-1552, 2008.

[28] A. Sinkar and N. Kim. Analyzing Potential Power Reduction with Adaptive Voltage Positioning Optimized for Multicore Processors. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED),* 2009.

[29] X. Qi and D. Zhu. Power Management for Real-Time Embedded Systems on Block-Partitioned Multicore Platforms. In *Proceedings of the Intl. Conf. on Embedded Software and Systems (ICESS),* 2008.

[30] C-Y. Yang, J-J. Chen, T-W. Kuo, and L. Thiele. An Approximation Scheme for Energy-Efficient Scheduling of Real-Time Tasks in Heterogeneous Multiprocessor Systems. In *Proceedings of ACM/IEEE Conference of Design, Automation, and Test in Europe (DATE),* 2009.

[31] C. Yang, J. Chen, and T-W. Kuo. An Approximation Algorithm for Energy-Efficient Scheduling on A Chip Multiprocessor. In *Proc. of the Conf. on Design, Automation and Test in Europe (DATE),* 2005.

[32] D. Zhu, R. Melhem and B. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems. In *IEEE Transactions on Parallel and Distributed Systems,* vol. 14, no. 7, pp. 686 - 700, 2003.

[33] D. Zhu, R. Melhem and D. Mosse. The Effects of Energy Management on Reliability in Real-Time Embedded Systems. In *Proc. of IEEE/ACM Intl. Conf. on Computer Aided Design (ICCAD),* 2004.

[34] http://www.acpi.info.