Utilization-controlled Task Consolidation for Power Optimization in Multi-Core Real-Time Systems

Xing Fu and Xiaorui Wang
Dept. of Electrical Engineering and Computer Science, University of Tennessee, Knoxville

***Abstract*—Since multi-core processors have become a primary trend in processor development, new scheduling algorithms are needed to minimize power consumption while achieving the desired timeliness guarantees for multi-core (and many-core) real-time embedded systems. Although various power/energy-efficient scheduling algorithms have recently been proposed, existing studies may have degraded run-time performance in terms of power/energy efficiency and real-time guarantees when applied to real-time embedded systems with uncertain execution times. In this paper, we propose a novel online solution that integrates core-level feedback control with processor-level optimization to minimize both the dynamic and leakage power consumption of a multi-core real-time embedded system. Our solution monitors the utilization of each CPU core in the system and dynamically responds to unpredictable execution time variations by conducting per-core voltage and frequency scaling. We then perform task consolidation on a longer timescale and shut down unused cores for maximized power savings. Both empirical results on a hardware multi-core testbed with well-known benchmarks and simulation results in many-core systems show that our solution provides the desired real-time guarantees while achieving more power savings than three state-of-the-art algorithms.**

I. Introduction

Multi-core processors have become a primary trend in the current processor development due to well-known technological barriers such as the "Power Wall" and "Instruction-level Parallelism Wall". As a result, future high-performance real-time embedded systems are anticipated to be equipped with multi-core processors, or even many-core processors (i.e., processors with tens or hundreds of cores). However, power consumption still remains the major constraint for the further throughput improvement of multi-core processors. For example, the peak power consumption of a multi-core processor often needs to be capped as high power consumption may result in high die temperatures that can affect the reliability and performance of the processor [28]. The power problem is also exacerbated by the rapidly increasing level of core integration in the multi-core processor design. Therefore, new scheduling algorithms must be developed to minimize power consumption while achieving the desired timeliness guarantees for multi-core (and many-core) real-time embedded systems.

Although various power/energy-efficient scheduling algorithms have recently been proposed for multi-core real-time embedded systems (e.g., [25]), existing studies focus on open-loop solutions such as *static* speed scheduling and offline DVFS (dynamic voltage and frequency scaling) configurations. For example, many existing speed scheduling algorithms optimize energy/power based on worst-case execution times (WCETs) and thus, may fail the optimization goal at runtime as the actual execution times can be much smaller than the WCETs. Recent projects propose scheduling algorithms with the assumption that execution times follow certain probability distributions (e.g., [10]). While those open-loop solutions can work effectively for traditional real-time embedded systems deployed in closed execution environments, they may incur degraded performance in terms of power/energy efficiency and real-time guarantees when applied to real-time embedded systems that execute in open and *unpredictable* environments in which workloads (*e.g.*, WCETs) are unknown and may vary significantly at runtime. Therefore, in order to achieve runtime power optimization and real-time guarantees, novel online strategies must be designed to dynamically respond to execution time variations for multi-core real-time embedded systems running in unpredictable environments.

Recently, feedback control techniques have been demonstrated to be a valid tool in providing timeliness guarantees for real-time embedded systems by adapting to workload variations based on dynamic feedback. In particular, feedback-based *CPU utilization control* [20] has been shown to be an effective way of providing real-time guarantees for soft real-time systems. The goal of utilization control is to enforce appropriate schedulable utilization bounds on all CPU cores in a real-time embedded system, despite significant uncertainties in system workloads. As a result, utilization control can guarantee all the real-time deadlines of the system without accurate knowledge of the workload, such as task execution times. However, existing utilization control algorithms are not designed to provide online power minimization for multi-core real-time systems. A recent study [27] proposes a power-aware utilization control approach that adopts DVFS to achieve utilization control and power efficiency. While this solution can effectively reduce *dynamic* power consumption, it cannot minimize *static (leakage)* power consumption because it does not minimize the number of active CPU cores in response to workload variations. As chip feature sizes continue to shrink, it becomes increasingly important to minimize leakage power since leakage power consumption is becoming a major contributor to the total power consumption of a multi-core processor [16].

To minimize the number of active CPU cores, it is necessary to migrate tasks among the cores for consolidation. In traditional multiprocessor real-time systems, tasks are often assigned to processors in a static way, at design time, due to the large overheads of online task migrations. A key advantage of the shared L2 caches in many multi-core real-

time systems is that the overhead of migrating a task among cores is less than 40 microseconds, which is sufficiently small in many real systems [3][30]. This feature allows multi-core real-time systems to be more power-efficient since the leakage power consumption can be minimized by dynamic task consolidation. Although task migrations in multi-core processors may cause L1 cache misses, the typical penalty of an L1 cache miss is only 10-30 CPU cycles. In contrast, in traditional multiprocessor real-time systems, task migrations can be expensive by having frequent L2 cache misses, whose penalty is approximately 100-300 CPU cycles [3].

In this paper, we propose a novel *online* solution that integrates feedback control with optimization strategies to minimize (both dynamic and leakage) power consumption and guarantee timeliness for multi-core real-time embedded systems. Our solution monitors the utilization of each CPU core in the system and dynamically responds to execution time variations by conducting per-core DVFS and task consolidation among the cores in a multi-core processor. In our solution, each CPU core has a utilization controller that throttles the DVFS level of the core so that its utilization stays slightly below the schedulable bound for minimized dynamic power with real-time guarantees. To minimize leakage power, we dynamically consolidate real-time tasks onto a few of the most power-efficient cores on a longer timescale by utilizing the small overhead of migrating tasks among different cores within a multi-core processor. The migration is subject to the schedulable utilization bounds of the active cores. We then shut down unused CPU cores for minimized leakage power.

The main contributions of this paper are three-fold:

- We propose a control theoretic solution for timeliness guarantees that minimizes both dynamic and leakage power consumption. Compared with traditional open-loop solutions, our solution can achieve better power efficiency and real-time performance when task execution times vary significantly at runtime in unpredictable environments.
- We design a two-level power optimization architecture that analytically integrates core-level utilization control with processor-level task consolidation to eliminate the complexity of one-level hybrid model-predictive control. The task consolidation problem is formulated as a bin-packing problem and several solutions are comparatively studied.
- While the majority existing work relies solely on simulations for evaluation, we present empirical results on a hardware multi-core testbed to demonstrate the efficacy of our integrated solution with the Mibench benchmarks [15]. Extensive simulation results also show that our solution can achieve more power savings than state-of-the-art algorithms in many-core systems.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents the integrated system architecture. Section IV introduces the core-level utilization control loop while Section V discusses the processor-level online task consolidation strategy. Section VI introduces the implementation of the integrated solution. Section VII presents our empirical results on our testbed and simulation results. Finally, Section VIII summarizes the paper.

II. Related Work

Several projects have addressed the scheduling problems for multi-core real-time embedded systems. Anderson et al. proposed a cache-aware scheduling technique to avoid cache thrashing for real-time tasks on multi-core platforms [3]. Guan et al. presented test conditions for non-preemptive EDF and fixed priority scheduling [14]. However, these studies do not migrate tasks for power optimization. Sarkar et al. studied the impact of task migrations on the WCETs of real-time tasks [24]. Their work focused on WCET analysis instead of real-time scheduling. In addition, they assume non-shared L2 caches which can incur a much higher task migration overhead. Chattopadhyay et al. studied the WCET analysis for a unified cache multi-core processors [7]. All of these studies do not address the power optimization problem in multi-core real-time systems. In contrast, we attempt to minimize the power consumption of multi-core real-time systems in addition to providing real-time guarantees.

Power management is an important problem for real-time embedded systems. Multiple projects have studied real-time scheduling with power management for uniprocessor systems (*e.g.*, [18]). Aydin et al. considered the energy-aware partitioning of real-time tasks for multiprocessor systems [5]. However, the power models of [5] did not consider leakage power consumption. Chen et al. extended the power models adopted in [5] and proposed a real-time scheduling method that minimizes both dynamic and leakage energy consumption [9]. However, these studies focus on multi-processor real-time systems where task migration can be expensive due to state maintenance. Seo et al. studied energy efficient multi-core real-time scheduling [25]. Their assumption is that all cores must run at the same frequency (chip-wide DVFS). In contrast, we utilize the availability of per-core DVFS for further power savings. As a result, the problem formulation is significantly different. All the aforementioned studies assume that task execution times are known a priori. While these studies can optimize the system power consumption when execution times do not change dynamically, the optimality is not guaranteed under execution time variations. Although much work on feedback control scheduling exists, to the best of our knowledge, our work is the first one which integrates the utilization control with DPM. Recently, [12] proposed to dynamically partition shared last-level caches of multi-core processors to control the utilization while reduce the power consumption. [12] is complementary to this work and can be integrated to further reduce the power consumption while guarantee real-time.

III. System Architecture

In this section, we present our system architecture. As shown in Figure 1, our system architecture features a task

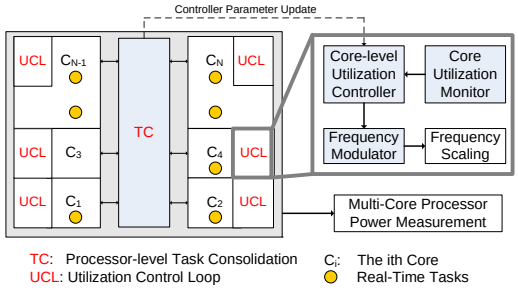


Fig. 1. System architecture

consolidation manager for the entire multi-core processor and a utilization control loop for each core in the processor.

First, for every core, a utilization controller exists that controls the CPU utilization of the core by scaling the core frequency. The controller is a Single-Input-Single-Output (SISO) controller since the change of core frequency only affects the utilization of the core. This control loop works as follows: (1) the utilization monitor on each core sends the utilization of the core to the local controller; (2) the controller computes a new CPU frequency and sends it to the frequency modulator on the core; and (3) the frequency modulator then changes the core frequency using DVFS.

Second, the processor-level task consolidation manager dynamically allocates tasks among the cores for task consolidation. It works as follows: (1) the task consolidation manager monitors all the tasks $\{T_i|1 \leq i \leq m\}$ and measures their CPU utilizations at run-time; (2) the task consolidation manager computes an optimized new task allocation for the cores and sends the task migration requests to the operating system. The OS then redistributes the following releases (i.e., jobs) of the periodic tasks to the cores to enforce the migration of the periodic tasks; and (3) the OS changes the affinity of the tasks to the cores accordingly. The overhead of migrating a task among cores is less than 40 μs which is sufficiently small in the majority of practical real-time systems. The detailed overhead measurement results can be found in [30].

In a real system, similar to the power management unit implemented in POWER7 [29], our control architecture can be implemented in service processor firmware to interact with the main processor and OS. Our solution can also be implemented in the OS as a periodic task with the highest priority. It is important to note that without effective integration, the processor-level task consolidation manager and the core-level utilization control loops may conflict with each other. The task consolidation manager may cause the core-level utilization control loop to be unstable, as it will change the system models used by the utilization controllers. As a result, the utilization control loops need to be configured with the proper controller parameters, according to task migration. One of our main contributions is to solve the design challenges when integrating processor-level task migration and core-level utilization control. We discuss the details of the integration in Section V-C.

IV. Core-level Utilization Control

In this section, we model, design, and analyze the core-level utilization control loop.

A. Task Model

To maximize the throughput of a multi-core system, an application assigned to run on multi-core processors typically consists of multiple tasks running in parallel; thus, we adopt a commonly used independent periodic task model (*e.g.*, in [3]). A system is comprised of m periodic tasks $\{T_i|1 \leq i \leq m\}$ executing on n cores $\{C_i|1 \leq i \leq n\}$ in a multi-core processor. Task T_i can be migrated among different cores. A core may host one or more tasks. Each task T_i has a *soft* deadline that is equal to its period. r_i is the inverse of the period of task T_i. A well-known approach for meeting the deadlines on a core is to ensure its CPU utilization remains below its schedulable utilization bound (e.g., Liu and Layland bound for RMS scheduling)[19]. Note that our task model can be extended to support aperiodic tasks by using the corresponding schedulable utilization bound. For example, a utilization bound has been derived for systems with aperiodic tasks in [2]. Task rate adaptation can also be used for utilization control in some real-time systems [20]. We focus on DVFS and task migration for a more general solution since the rates of many real-time tasks cannot be adapted.

Our task model has two important properties. First, while each task T_i has an *estimated* execution time c_i available at design time, a real-time task's *actual* execution time may differ from its estimation and vary at run-time due to two reasons: core frequency scaling by the DVFS and workload uncertainties. Modeling such uncertainties is important to systems operating in unpredictable environments. The estimated execution time can be an approximate estimation and is not necessarily the WCET. Second, the core frequency of each core C_i can be dynamically adjusted on a per-core basis within a range $[F_{min}, F_{max}]$. This assumption is based on the fact that more energy savings can be achieved with per-core DVFS when compared to conventional chip-wide DVFS [17] and many today's microprocessors already support per-core DVFS (e.g., AMD Independent Dynamic Core Technology). Note that our solution does not rely on WCET estimation, which is a key advantage of our solution, because WCETs are often unavailable or mis estimated in real-time embedded systems running in open execution environments. In contrast, a fundamental limitation of open-loop power optimization solutions is that they may fail the optimization goal at runtime when the actual execution times are significantly different from the WCETs used in the optimization. The frequency ranges are assumed to be continuous because a continuous value can be approximated by a series of discrete frequency levels supported by a processor, as we explain in Section VI.

B. System Modeling

We first introduce the following notation. T_s, the control period, is selected such that multiple jobs of each task may be released during a control period. The utilization control loop is invoked every T_s seconds. $u_i(k)$ is the utilization of core C_i in the k^{th} control period, i.e., the fraction of time that C_i is not idle during the time interval $[(k-1)T_s, kT_s)$. B_i is the desired utilization set point (*i.e.*, schedulable bound) on C_i.

$S_i(k)$ is the set of tasks located on core C_i in the k^{th} control period. $f_i(k)$ is the normalized CPU frequency (*i.e.*, a value relative to the highest level F_{max}) of core C_i in the k^{th} control period.

Following a control-theoretic methodology, we establish a dynamic model that characterizes the relationship between the controlled variable $u_i(k)$ and the manipulated variables $S_i(k)$ and $f_i(k)$. As observed in [23], since the frequencies of real microprocessors can be scaled only within limited ranges, the execution times of computation-intensive tasks on core C_i can be approximately estimated to be proportional to C_i's relative core frequency[1]. Therefore, when core C_i runs at $f_i(k)$, the *estimated* execution time of task T_i on C_i in the k^{th} control period can be modeled as $c_i/f_i(k)$. The *estimated* CPU utilization of core C_i can be modeled as

$$b_i(k) = \frac{\sum_{T_j \in S_i(k)} c_j r_j}{f_i(k)} \tag{1}$$

We then define the *estimated* utilization change of C_i, $\Delta b_i(k)$, as

$$\Delta b_i(k) = \frac{\sum_{T_j \in S_i(k+1)} c_j r_j}{f_i(k+1)} - \frac{\sum_{T_j \in S_i(k)} c_j r_j}{f_i(k)}. \tag{2}$$

Note $\Delta b_i(k)$ is based on the estimated execution time c_j. Since the actual execution times may differ from their estimation due to workload variations, we model the actual utilization of C_i as the following difference equation

$$u_i(k+1) = u_i(k) + g_i \Delta b_i(k) \tag{3}$$

where the utilization gain g_i represents the ratio between the change to the actual utilization and its estimation $\Delta b_i(k)$. For example, $g_i = 2$ means that the actual change to utilization is twice the estimated change. Also note that the exact value of g_i is *unknown* at design time due to the unpredictability of the tasks' execution times.

The system models (2) and (3) show the actual utilization determined by both the frequency and task allocation. Since $S_i(k)$ contains a discrete number of tasks, the system model introduces a significant challenge, which usually requires *hybrid model-predictive control* [21]. In a model-predictive controller, the control problem is translated to a constrained least-squares problem [20]. The hybrid model-predictive control problem is translated to a mixed integer non-linear programming problem (MINLP) and all existing MINLP solvers are not polynomial algorithms.

To address this challenge, we adopt an integrated optimization and control approach. First, we determine the task allocation based on an optimization strategy introduced in Section V. The goal is to minimize the power consumption of the multi-core system. Second, a feedback controller is designed for each core to achieve the desired utilization. Based on our control architecture, the core-level utilization control loop can be designed separately from the task migration optimization strategy. As a result, model (3) can be simplified by having $S_i(k)$ in (2) as a constant S_i. This avoids designing a controller based on (2) to handle discrete changes of the task allocation. As a result, model (3) becomes

$$u_i(k+1) = u_i(k) + g_i \Delta d_i(k) \sum_{T_j \in S_i} c_j r_j \tag{4}$$

where $\Delta d_i(k) = 1/f_i(k+1) - 1/f_i(k)$. The model cannot be directly used to design the controller because the system gain g_i is used to model the uncertainties in task execution times and is unknown at design time. Therefore, we design the controller based on an approximate system model of (4) with $g_i = 1$. In a real system where the task execution times differ from their estimations, the *actual* value of g_i may not equal 1. As a result, the closed-loop system may behave differently. However, we show that a system controlled by a controller designed with $g_i = 1$ can remain stable when the variation of g_i is within a certain range. This range is established using a stability analysis of the closed-loop system by considering model variations.

C. Controller Design and Analysis

Because of our novel control architecture, the model (3) is simplified as the model (4), and we can borrow the controller design in [27]. The Z-transform of the P controller [27] is

$$C(z) = \frac{1}{\sum_{T_j \in S_i} c_j r_j}. \tag{5}$$

The transfer function of the closed-loop system controlled by controller (5) is

$$G(z) = z^{-1}. \tag{6}$$

It is easy to prove that the controlled system is stable and has zero steady state errors when $g_i = 1$. When the designed P controller is used on a system with $g_i \neq 1$, the system will remain stable when $0 < g_i < 2$, which means that the actual utilization change *cannot* be twice the estimated utilization change. We have also proven that the system can achieve zero steady state error when the system is stable.

V. Processor-level Task Consolidation

In this section, we first formulate the problem of power optimization with uncertain execution times. Second, we show that the problem is an NP-complete problem and investigate four heuristics. Finally, we analyze the coordination between the processor-level task consolidation and core-level utilization control.

A. Problem Formulation

In addition to notation defined in Section IV-B, we introduce more notation. T_{op} is the time interval between two consecutive optimization invocations. The value of T_{op} can be selected based on the trade-off between the system response speed to workload variations and the overhead of task migration in the system. T_{op} is normally longer than T_s such that the

[1]In general, the execution times of tasks which have intensive memory access and I/O operations may include frequency-independent parts that do not scale proportionally with the core frequency [4]. We plan to model frequency-independent parts in our future work.

core-level utilization control loops can settle down within two consecutive invocations of the task consolidation algorithm.

In multi-core systems, the processor power consumption $P(k)$ is commonly modeled [22] as follows

$$P(k) = P_s + \sum_{i=1}^{n} x_i(k)[P_{ind}^i + P_d^i(k)] \tag{7}$$

where P_s denotes the static power of all power consuming components(except the cores). This is approximately a constant and can only be removed by powering off the entire processor. x_i represents the state of core C_i. If a core is active, $x_i = 1$; otherwise, $x_i = 0$ and C_i is turned off. The frequency-dependent active power for the core is defined as $P_d^i(k)$=$\alpha_i f_i(k)^{\beta_i}$, where both α_i and β_i are system-dependent parameters. P_{ind}^i is the static power of Core i and does not depend on the supply voltage and frequency.

Given a utilization set-point vector, $\mathbf{B} = [B_1 \dots B_n]^T$ and frequency constraints $[F_{min}, F_{max}]$ for each core C_i, the optimization goal at the k^{th} point (time kT_{op}) is to dynamically choose a task allocation $\{S_i(k)|1 \leq i \leq n\}$ and core frequencies $\{f_i(k)|1 \leq i \leq n\}$ to minimize the processor power consumption $P(k)$

$$\min \sum_{i=1}^{n} x_i(k)[P_{ind}^i + \alpha_i f_i(k)^{\beta_i}] \tag{8}$$

where $x_i(k)$ is defined as a function of $S_i(k)$

$$x_i(k) = \begin{cases} 0, & S_i(k) = \phi \\ 1, & S_i(k) \neq \phi \end{cases} \tag{9}$$

subject to the following constraint

$$\sum_{T_j \in S_i(k)} U_j \leq B_i \tag{10}$$

where U_j is the average CPU utilization of task T_j which can be obtained from the operating system.

Constraint (10) ensures that the aggregate utilization of all the tasks on core C_i is smaller than the schedulable utilization bounds [19]. As a result, all the tasks on each core can meet their real-time deadlines. $f_i(k)$ in (8) can also be computed as a function of $S_i(k)$ based on the following equation

$$f_i(k) = \frac{\sum_{T_j \in S_i(k)} U_j}{B_i}. \tag{11}$$

Note that $f_i(k) \leq 1$ is enforced by constraint (10). If $f_i(k) \leq F_{\min}$, we set $f_i(k) = F_{\min}$ and the real-time guarantees will not be violated because the core frequency is higher than required.

Therefore, the design goal of the task consolidation algorithm is to determine a task allocation $S_i(k)$ that can minimize power consumption $P(k)$. Task consolidation and idle core shutdown can lead to more power savings than when simply using DVFS to lower core frequencies because as feature sizes decrease below 65 nm, the leakage power consumption becomes a major contributor to the total power consumption of a processor [16][26]. For example, in 23nm processors, the leakage power consumption accounts for approximately 80% of the total power consumption. For multi-core and many-core systems, the leakage power of idle cores can constitute a significant portion of the total power. The experiments on our hardware testbed (as shown in Figure 2) also demonstrate that the task consolidation and idle core shutdown result in more power savings than a DVFS-only solution.

Clearly, the optimization problem formulated in (8) for dynamic task consolidation can be transformed to a bin packing problem [19] as it needs to pack all tasks to the cores based on their CPU utilizations and the capacity of a core shrinks when the number of its tasks increases. In the following, we discuss solutions to the formulated bin-packing problem.

B. Heuristics

Since the bin-packing problem is known to be NP-complete and so an optimal solution is not suitable to be used online in multi-core and many-core systems with many tasks, most existing work focuses on heuristics. Several suboptimal heuristics with different complexities have been proposed. In this work, we evaluate and compare several heuristics in terms of both overhead and solution quality. Note that for real-time embedded systems, run-time overhead is often a more serious concern than solution quality. High run-time overheads may impact the schedulibility of real-time tasks and cause deadline misses.

We test First-Fit, Best-Fit, and an advanced bin packing heuristic rt-MBS based on MBS (Minimum Bin Slack) [11]. To further reduce the overhead of First-Fit, we design *iFF* (Incremental First-Fit). In this section, we will compare the overheads of all heuristics theoretically. In Section VII-B, we evaluate four different heuristics presented in Section V-B in terms of both overhead and solution quality using realistic workloads.

First-Fit places each task, in succession, into the first core into which it fits. Best-Fit places each task, in succession, into the most nearly full core in which it fits. Incremental First-Fit has two arrays to hold the task allocation in the last control period and task allocation in the current control period, respectively. Incremental First-Fit also employ First-Fit to assign each task into a core in every control period. However, in contrast to that First-Fit assign every task into a core by calling a system call, Incremental First-Fit store the assignment of every task into a core in an array instead of calling the system call immediately, then compare the new task allocation with the task allocation in the last control period and only call the system call for the task with changed core affinity. The key observation of iFF is that the order in which tasks are packed into a core is irrelevant. What is important is the total number of core and the total utilizations of each core are the same as First-Fit. The system call overhead is up to 40 microseconds [30]. For every task, First-Fit needs to call the system call once. For a large of number tasks, the overhead may be big. Incremental First-Fit eliminates those unnecessary system calls and provide the same solution quality as First-Fit.

MBS is bin-focused. In each step, MBS attempts to find a set of tasks (packing) that makes the core as full as possible.

Algorithm 1 rt-MBS

q: an index of tasks not assigned to cores
n: the number of tasks not assigned to cores
A: the current assignment
Minimum-Bin-Slack (q)
begin
1: **for all** index i from q to n **do**
2: Get i^{th} tasks not assigned to cores;
3: **if** i^{th} tasks can be assigned to the core under A **then**
4: Add i^{th} tasks into A;
5: Minimum-Bin-Slack($i+1$);
6: Remove i^{th} tasks from A;
7: **if** No free space exists under the current optimal assignment **then**
8: Exit;
9: **end if**
10: **end if**
11: **if** A is better than the current optimal assignment **then**
12: Set A the current optimal assignment;
13: **end if**
14: **end for**
end

Building a packing for each core is implemented recursively. The detailed algorithm of applying MBS to processor-level task consolidation is shown in Algorithm 1. The algorithm is invoked repeatedly until all tasks are assigned. The procedure is invoked with $q = 1$ while the current assignment and current optimal assignment are initialized to be null sets. Note that the allocation in each step is subject to the utilization constraint (10), which is enforced by line 3 of Algorithm 1. The utilization constraint is checked in each step when a task is allocated to a core to guarantee the real-time executions of tasks.

We now analyze the complexity of the four heuristics. First-Fit and Best-Fit are among the simplest heuristics. MBS, in the worst case, has the same complexity as an exhaustive search. The complexity of Incremental First-Fit, First-Fit, Best-Fit, and MBS is $O(mlogm)$, $O(mlogm)$, $O(mlogm)$, and $O(m^{u+1})$, respectively; where m is the total number of tasks in the system and u is the maximum number of tasks that can be placed in one core. The overhead of Incremental First-Fit is smaller than that of First-Fit because of fewer system calls. The improved time complexity is archived by using two more arrays with space complexity of $O(m)$.

C. Integration of Optimization and Feedback Control

We now analyze the integration needed for the core-level utilization control loops to work with task consolidation.

Specifically, we need to ensure that the stability of the core-level utilization control is not affected when the processor-level task consolidation reallocates the tasks to each core. Equation (5) reveals that the controller parameter is determined by the task set on the core. We now analyze the stability of the core-level utilization control when the controller parameter is incorrectly configured. First we define γ to be

$$\gamma = \frac{\sum_{T_j \in X_i} c_j r_j}{\sum_{T_j \in S_i} c_j r_j} \tag{12}$$

where $\sum_{T_j \in S_i} c_j r_j$ is the utilization controller parameter and X_i is the task allocation determined by the processor-level task consolidation. The transfer function of the closed-loop system controlled by the controller (6) becomes:

$$G(z) = \frac{\gamma}{z - (1-\gamma)}. \tag{13}$$

From (13), if $|1-\gamma|$ is less than 1, the system will be stable since the pole is inside the unit circle in the Z-plane.

The processor-level task consolidation manager will check task migrations against the derived stability criteria. If the system would become unstable after the migration, the task migration will be prohibited.

VI. System Implementation

We first introduce the physical testbed used in our experiments. Next we introduce our simulation environment.

A. Physical Testbed

Our testbed is an Intel Xeon X5365 Quad Core processor with an 8MB on-die L2 cache and 1,333 MHz Front Side Bus. The processor supports four DVFS levels: 3GHz, 2.67GHz, 2.33GHz, and 2GHz. According to Intel, the processor has Core 0 and Core 1 fabricated on one die and Core 2 and Core 3 on a separate die. We must change the DVFS levels of the 2 cores on each die in order to have a real impact on the processor power consumption. Therefore, we use this processor to emulate a dual-core processor that supports a per-core DVFS. The operating system is a Fedora Core 7 with a Linux kernel 2.6.23 and a real-time-preempt kernel patch.

The default Linux kernel may migrate real-time tasks by itself, which can cause deadline misses as the core utilizations are not guaranteed by the kernel during migration. To disable task migration from the Linux kernel, we use a standard system call SCHED_SETAFFINITY [6], which is a portable approach across different platforms. The overhead of the system call for task migration among cores is less than 40 μs which is acceptable in many real-time embedded systems. The detailed overhead results are in [30].

We adopt the Mibench benchmarks [15] designed for embedded systems as our tasks. Our experiments run a medium-sized workload comprised of 10 tasks to run the Mibench benchmarks. Both cores initially have five periodic tasks with a total utilization of 0.31. The task parameters such as periods are configured according to a real real-time application [1]. The tasks on each core are scheduled by the RMS algorithm [19]. Note that our solution can also be used with other scheduling approaches, such as EDF, as long as the corresponding schedulable utilization bound is adopted. We use RMS as an example in this paper because RMS usually has a smaller runtime overhead in real systems. The deadline of

each task T_i equals its period, $1/r_i$. The utilization set point of every core is set to its RMS schedulable utilization bound [19], i.e., $B_i = m(2^{1/m} - 1)$, where m is the number of tasks on C_i. Since the number of tasks may change according to the processor-level task consolidation, the set point can be set to 0.69 which is the limit of $B_i = m(2^{1/m} - 1)$ when $m \to \infty$. All tasks meet their deadlines if the desired utilization on every core is enforced.

We now introduce the implementation details of each component in our system architecture.

Utilization Monitor: The utilization monitor uses the /proc/stat file in Linux to estimate the core utilization in each sampling period. The /proc/stat file records the number of jiffies (usually 1ms - 10ms in Linux) when a core is in user mode, user mode with low priority (nice), system mode, and when used by the idle task, since the system starts. The utilization of each task can be calculated based on the number of jiffies consumed by the task process in each control period.

Core-level Utilization Controller: The controller is implemented as a single-thread process with the highest priority running on each core. With a control period of 30 second, the controller periodically reads the core utilization, executes the control algorithm presented in Section IV-C to compute the desired core frequency, and sends the new frequency to the frequency modulator on the core.

Frequency Modulator: We use Intel's Enhanced SpeedStep Technology to enforce the new CPU frequency. To the change core frequency, one needs to install the *cpufreq* package and then use the root privilege to write the new frequency level into the system file. A routine periodically checks this file and resets the core frequency accordingly. The average overhead (*i.e.*, transition latency) to change the frequency in Intel processors is approximately $100\mu s$.

Power Monitor: To measure the power consumption of the processor, an Agilent 34410A digital multimeter (DMM) is used with a Fluke i410 current probe to measure the current running through the 12V power lines that power the processor. The probe is clamped to the 12V lines and produces a voltage signal proportional to the current running through the lines with a coefficient of 1mv/A. The resultant voltage signal is then measured with the multi-meter. The accuracy of the probe is 1.5% of reading + 0.5A.

B. Simulation Environment

Our simulation environment is composed of an event-driven simulator implemented in 3,546 lines of C++ code. The simulator implements a multi-core real-time system with our control architecture, utilization monitor, frequency modulator, and task consolidation manager.

The synthetic real-time applications are randomly generated with each task set containing 80-640 tasks, where their initial execution times are randomly generated following a uniform distribution. The task period of each task is also randomly generated within a range.

In our simulations, we consider processors with 16, 32, 64, 128, and 256 cores. The power model parameters used in our simulations are based on the power models of many-core processors used in [16][8]. For the parameters in the power model (8), we set $\alpha_i = 1, \beta_i = 3$ and a normalized frequency is used with $F_{max} = 1$. The normalized maximum frequency-dependent active power on each core is $P_d^i = 1$. Moreover, the static power is configured as $P_s = 0.01$ normalized with respect to P_d^i. According to [16], as feature sizes shrink below 0.1 micron, P_{ind}^i can become comparable to P_d^i in the near future. Thus, the normalized frequency-independent active power for one core is configured to be $P_{ind}^i = 1$. The power model is implemented using numerical methods.

VII. Evaluation

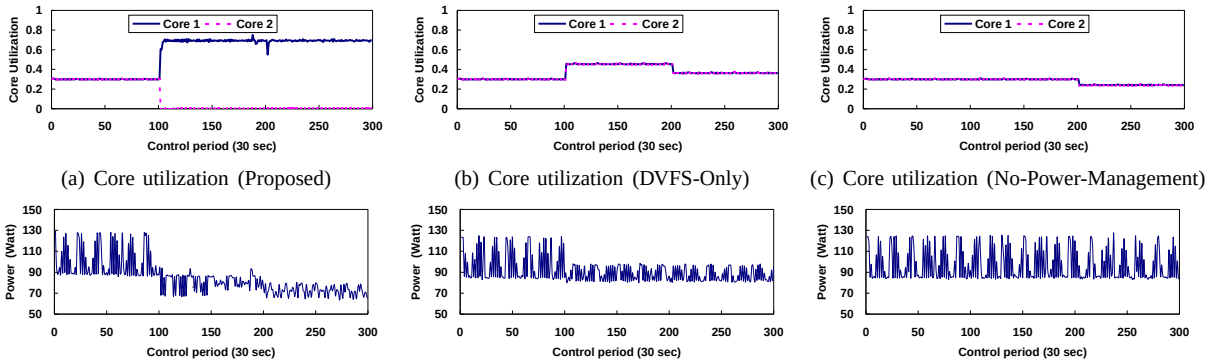
In this section, we first compare four heuristics in Section V-B, then present our empirical results conducted on the hardware multi-core testbed. To stress test our solution, we finally describe our simulation results in many-core systems.

A. Baselines

We use three baselines for comparison in this paper. *Dynamic core scaling* is a state-of-the-art algorithm [25], which adjusts both the core frequencies and number of active cores of a multi-core system to reduce the dynamic and leakage power consumption by task migration. The fundamental difference between Dynamic core scaling and the proposed solution is that the Dynamic core scaling makes a task migration decision based on the WCET of the task to be migrated. For systems operating in unpredictable environments, to guarantee the timeliness, the WCETs have to be conservative. The actual execution time of the task to be migrated is usually much smaller than the overestimated WCET. In contrast, the proposed solution makes a task migration decision based on the average CPU utilization, which can be easily monitored at runtime in a lightweight way. In addition, Dynamic core scaling uses a chip-wide DVFS while the proposed solution uses a per-core DVFS, which is already supported by many microprocessors. We demonstrate in Section VII-D that the proposed solution outperforms Dynamic core scaling significantly in terms of power savings. The second baseline, *DVFS-Only*, is the frequency scaling loop proposed in [27]. It relies only on DVFS to throttle the core frequency to manage the power consumption of a core, subject to the utilization constraints without turning off any cores. DVFS-Only has a similar utilization controller design with the proposed solution, but does not perform task consolidation. *No-Power-Management* is a classical open-loop scheduling solution that partitions the tasks in a static way [19] and the frequencies of all cores in a multi-core system are fixed to the maximum frequency level. While No-Power-Management can initially guarantee the timeliness, it may fail when task execution times change at runtime and waste energy when the system is underutilized.

B. Comparison of Different Heuristics

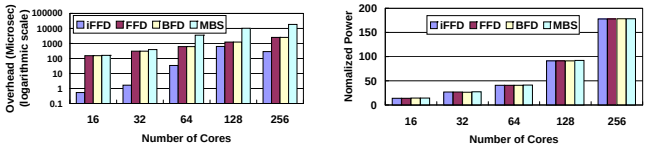
A scalability requirement for a multi-core or many-core power optimization heuristic is low run-time overhead. In this experiment, We evaluate four different heuristics presented in Section V-B in terms of both overhead and performance by simulations. Different workloads including 16 to 256 tasks are randomly generated to stress test all heuristics. To estimate the



(a) Core utilization (Proposed) (b) Core utilization (DVFS-Only) (c) Core utilization (No-Power-Management)

(d) Processor power consumption (Proposed) (e) Processor power consumption (DVFS-Only) (f) Processor power (No-Power-Management)

Fig. 2. Typical runs of three solutions (Proposed, DVFS-Only, and No-Power-Management) on the hardware testbed. The solutions are activated at the 100th control period and handle a 20% execution time reduction at the 200th control period.



(a) The overhead of four optimization heuristics (b) Power consumptions using four optimization heuristics

Fig. 3. Comparison among the three heuristics

overhead of the heuristics, we measure the execution time of each heuristic on a 2.5-GHz Intel Core 2 Duo PC with 2-GByte RAM. To obtain high-resolution measurement, we use Windows API *QueryPerformanceCounter*. We collect the average of multiple runs. As shown in Figure 3(a), the overhead of incremental First-Fit is smallest, while the overhead of MBS is significantly higher than the others. Figure 3(b) shows that under realistic workloads, the processor power consumption under all heuristics is very close. According to the simulations, we adopt incremental First-Fit for online power reduction in the following experiments because of its low overhead.

C. Empirical Results on Hardware Testbed

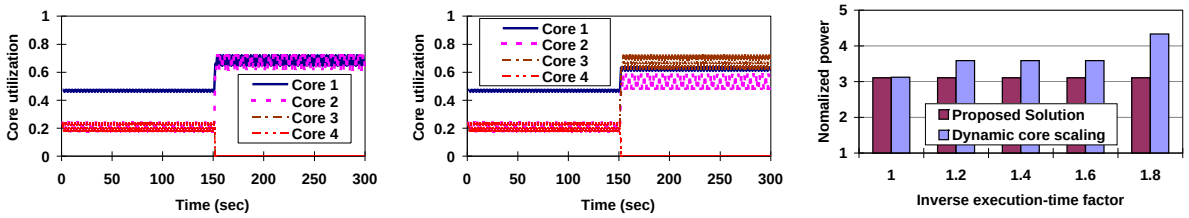
In this experiment, we first disable the proposed solution from the 1st to the 100th control period. As shown in Figure 2(a), the initial utilizations of Core 1 and Core 2 are both 0.31. Core utilizations are lower than the RMS bound, resulting in an undesired underutilized system. We then enable the proposed solution at the 100th control period. As shown in Figure 2(a), all tasks on Core 2 are migrated to Core 1. Core 2 becomes idle and is then turned off. As shown in Figure 2(d), the power consumption of the CPU is consequently reduced by approximately 19%. As shown in Figure 2(a), at the 200th control period, the execution time of all the tasks is suddenly decreased by 20%, resulting in a sharp drop of the utilization of Core 1. This decrease is implemented by reducing the number of loop iterations in the Mibench benchmarks. The proposed solution responds to the utilization drop by dynamically decreasing the core frequency of the core. Since the settling time of the utilization controller is just several control periods, the utilization converges quickly to the RMS bound again. As shown in Figure 2(d), the power consumption of the CPU is further reduced by approximately 11%. The experiment demonstrates the effectiveness of the proposed solution with uncertain task execution times.

We then examine the power efficiency of two baselines: DVFS-Only and No-Power-Management. To make a fair comparison, we adopt the same workload and scenario used for the proposed solution. For DVFS-Only, Figure 2(b) shows that at the 100 control period the utilization of all the cores increases because DVFS-Only throttles the frequencies of both cores to the lowest levels. As a result, Figure 2(e) shows the processor power drops at the 100th control period. However, the power consumption is still much higher than that of the proposed solution. The reason is that DVFS-Only cannot consolidate tasks to reduce the leakage power consumption of the processor. At the 200th control period, even though the execution times of all the tasks are decreased by 20%, DVFS-Only can only achieve very slightly further power savings because both the cores are already at their lowest frequencies. This experiment demonstrates the necessity of task consolidation. For No-Power-Management, as shown in Figure 2(c), at the 200th control period, the execution times of all tasks are decreased by 20%. Since No-Power-Management does not decrease the core frequencies in response to the lower workload, Figure 2(f) shows that the processor power is only slightly reduced and is much higher than that of the proposed solution. Since all the three solutions do not violate the RMS schedulable utilization bounds in their entire runs, no deadline miss is observed in this experiment for any of the solutions.

D. Simulation Results

In this section, we first compare the proposed solution with Dynamic core scaling on a quad-core system. We then test the effectiveness of the proposed solution in many-core systems. We have also performed the evaluation of the proposed solution in heterogeneous multi-core systems. The results are not presented due to space limitations but can be found in [13].

1) Comparison with Dynamic Core Scaling: In this section, we compare the power efficiency of the proposed solution and Dynamic core scaling in unpredictable environments. The WCETs of tasks often have to be conservative in unpredictable environments as the actual execution time may vary across in a wide range at runtime. The majority of the time, the actual



(a) Proposed solution (activated at the 150s) (b) Dynamic core scaling (activated at 150s) (c) Power consumption under different ietf

Fig. 4. Comparison between the proposed solution and Dynamic core scaling.

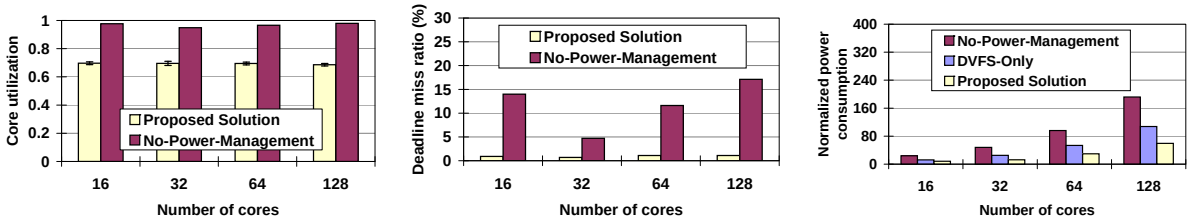
execution times can be much smaller than the pessimistic WCETs. The *inverse execution-time factor (ietf)* denotes the ratio of the estimated execution time to the actual execution time of a periodic task. The greater the *ietf* is, the more conservative the estimated execution time of a task. For Dynamic core scaling, the *ietf* can be determined by the predictability of the environment.

In the first experiment, we randomly generate a small scale task set including 5 tasks in a quad-core system. The *ietf* of the tasks is 1.5. Figure 4(a) shows that all tasks are consolidated onto two cores (Cores 1 and 2) under the proposed solution. In contrast, Figure 4(b) shows all tasks are consolidated onto *three* cores (Cores 1 to 3). The reason is that Dynamic core scaling relies on WCETs to decide whether or not it migrates a task. Because of the overestimated WCETs (*i.e.*, ietf=1.5), Dynamic core scaling may prevent task migrations. Dynamic core scaling cannot make task migration decisions based on actual execution times. Otherwise, schedulable bounds may be violated after migrations and deadline misses occur. In contrast, the proposed solution relies on the feedback of the average task CPU utilizations and so tasks can be consolidated onto fewer cores. Note that when the actual execution time of a task approaches the WCET, the proposed solution can still guarantee the timeliness by dynamically enforcing the schedulable utilization bounds. Figure 4(b) also shows that only Core 1 reaches the utilization bound under Dynamic core scaling due to its assumption of chip-wide DVFS. If the workload is not perfectly balanced, which is common in a real system, chip-wide DVFS cannot allow all cores to reach the RMS bound at the same time, resulting in undesired underutilized systems and unnecessarily more power consumption. In this experiment, after the activation at 150s, the normalized power consumption of the proposed solution is reduced from 8.01 to 3.106, while that of Dynamic core scaling is reduced from 8.01 to 3.587. Dynamic core scaling consumes about 15.5% more power than the proposed solution. The reason is that the proposed solution can consolidate tasks to reduce leakage power and utilize per-core DVFS to save more dynamic power.

We then compare the normalized power consumption of the proposed solution and Dynamic core scaling when the *ietf* varies from 1 to 1.8. Figure 4(c) shows when the ietf is 1, which means the the actual execution times are equal to the WCETs, the normalized power consumption of Dynamic core scaling is approximately the same as that of the proposed solution. The slight difference is because Dynamic core scaling does not utilize per-core DVFS. When the ietf increases from 1 to 1.2, the normalized power consumption of Dynamic core scaling increases to approximately 15.5% more than that of the proposed solution. The reason is that when the ietf is 1, both the solutions consolidate tasks onto two cores. When the task WCETs increase to 1.2 times of the actual execution times (*i.e.*, ietf=1.2), Dynamic core scaling uses three core while the proposed solution still uses only two. When the ietf increases from 1.2 to 1.6, the difference between the two solutions only changes marginally because Dynamic core scaling still utilizes three cores in this case. However, when the ietf further increases to 1.8, Dynamic core scaling begins to use all four cores. As a result, it consumes 39.6% more power than the proposed solution. Since both the proposed solution and Dynamic core scaling can enforce the CPU utilization dynamically on each core, the two solutions both achieve a zero deadline miss ratio in all runs. This experiment demonstrates that the proposed solution significantly improves the power efficiency of real-time systems in unpredictable environments.

2) Many-Core Systems: In this set of experiments, we evaluate the proposed solution in four many-core systems with 16, 32, 64, and 128 cores. The workload of each many-core system is randomly generated using the tool introduced in Section VI-B.

We first test the real-time performance under execution time uncertainty between the proposed solution and No-Power-Management, which is an open-loop solution and cannot adapt to runtime execution time variations. The utilization set points for all the cores is 0.69 (*i.e.*, the RMS bound). In the middle of each run, the execution times of all the tasks on C_1 are increased by 100%. As shown in Figure 5(a), the proposed solution effectively controls the CPU utilization of C_1 back to the set point and so, on average, has a zero steady state error with a maximal deviation less than 0.01. In contrast, the average utilization of C_1 scheduled by No-Power-Management is much higher than the set point due to the significant execution time increase at runtime. Figure 5(b) plots the deadline miss ratios of the two solutions. The deadline miss ratio is calculated as the total misses of all the tasks divided by the total number of released jobs. As shown in Figure 5(b), the deadline miss ratio of the proposed solution is much lower than that of No-Power-Management. The reason is that the utilization of C_1 under the proposed solution is controlled below the schedulable bound the majority of the time, except during the short time interval that the controller takes to control the utilization back to the set point after



(a) Average core utilizations with deviations (b) Deadline miss ratio (c) Processor power consumption

Fig. 5. Comparison of the proposed solution and baselines in many-core systems

the execution time increase. The temporary violation of the utilization bound results in some deadline misses. Since No-Power-Management is an open-loop solution and cannot adapt to run-time execution time variations, thus the utilization bound is violated significantly due to the execution times increases.

We then conduct another set of experiments to compare the proposed solution with No-Power-Management and DVFS-Only in terms of power consumption, without significant execution time variations tested in the first set of experiments. The proposed solution and DVFS-Only are activated at the beginning of each run. Figure 5(c) plots the normalized power consumption of the three solutions. The proposed solution has approximately 69% and 45% more power savings than No-Power-Management and DVFS-Only, respectively, when the number of cores is 128. The experiments demonstrate that the proposed solution can lead to significantly more power savings than No-Power-Management and DVFS-Only, while no deadline misses occur for all the three solutions.

VIII. Conclusions

Existing power/energy-efficient scheduling algorithms focus heavily on open-loop optimization solutions. As a result, they may have degraded run-time performance in terms of power/energy efficiency and real-time guarantees when they are applied to real-time embedded systems with uncertain execution times. In this paper, we have presented a novel online solution that integrates core-level feedback control with a processor-level optimization strategy to minimize both the dynamic and leakage power consumption of a multi-core real-time embedded system. Our solution monitors the utilization of each CPU core in the system and dynamically responds to unpredictable execution time variations by conducting per-core DVFS. Our solution then takes advantage of the small overhead of task migration in multi-core processors with shared L2 caches to perform task consolidation on a longer timescale and shuts down unused cores for maximized power savings. Both empirical results on a hardware multi-core testbed with the Mibench benchmarks and simulation results in many-core systems show that our solution provides the desired real-time guarantees while achieving more power savings than state-of-the-art algorithms.

References

[1] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. QoS negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11):1170–1183, 2000.

[2] T. F. Abdelzaher, V. Sharma, and C. Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers*, 53(3):334–350, 2004.

[3] J. H. Anderson, J. M. Calandrino, and U. C. Devi. Real-time scheduling on multicore platforms. In *RTAS*, 2006.

[4] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *RTSS*, 2006.

[5] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *IPDPS*, 2003.

[6] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition*. O'Reilly Publishers, 2005.

[7] S. Chattopadhyay and A. Roychoudhury. Unified cache modeling for wcet analysis and layout optimizations. In *RTSS*, 2009.

[8] J. Chen, S. Wang, and L. Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. In *RTAS*, 2009.

[9] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS*, 2006.

[10] P. Eles, Z. Peng, and S. Manolache. Schedulability Analysis of Applications with Stochastic Task Execution Times. *ACM Transactions on Embedded Computing Systems*, Nov. 2004.

[11] K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 2002.

[12] X. Fu, K. Kabir, and X. Wang. Cache-aware utilization control for energy efficiency in multi-core real-time systems. In *ECRTS*, 2011.

[13] X. Fu and X. Wang. Utilization-controlled Task Consolidation for Power Optimization in Multi-Core Real-Time Systems, Tech Report. http://www.ece.utk.edu/∼xwang/papers/Multicore-RT.pdf, 2010.

[14] N. Guan et al. New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms. In *RTSS*, 2008.

[15] M. R. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *WCC*, 2001.

[16] N. S. Kim et al. Leakage current: Moore's law meets static power. *Computer*, Dec. 2003.

[17] W. Kim et al. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, 2008.

[18] M. Kondo and H. Nakamura. Dynamic processor throttling for power efficient computations. In *PACS*, 2004.

[19] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[20] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Transactions on Parallel and Distributed Systems*, 16(6), 2005.

[21] J. Lunze and F. Lamnabhi-Lagarrigue. *Handbook of Hybrid Systems Control: Theory, Tools, Applications*. Cambridge University Press, 2009.

[22] X. Qi and D. Zhu. Power management for real-time embedded systems on block-partitioned multicore platforms. In *ICESS*, 2008.

[23] S. Saewong and R. R. Rajkumar. Practical voltage-scaling for fixed-priority RT-systems. In *RTAS*, 2003.

[24] A. Sarkar, F. Mueller, H. Ramaprasad, and S. Mohan. Push-assisted migration of real-time tasks in multi-core processors. In *LCTES*, 2009.

[25] E. Seo, J. Jeong, S. Park, and J. Lee. Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors. *IEEE Transactions on Parallel and Distributed Systems*, Nov. 2008.

[26] X. Tang, H. Zhou, and P. Banerjee. Leakage power optimization with dual-vth library in high-level synthesis. In *DAC*, 2005.

[27] X. Wang, X. Fu, X. Liu, and Z. Gu. Power-aware cpu utilization control for distributed real-time systems. In *RTAS*, 2009.

[28] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA*, 2009.

[29] M. Ware et al. Architecting for power management: The IBM POWER7 approach. In *HPCA*, 2008.

[30] Y. Zhang, C. Gill, and C. Lu. Real-time performance and middleware for multiprocessor and multicore linux platforms. In *RTCSA*, 2009.