Cache-Aware Utilization Control for Energy Efficiency in Multi-Core Real-Time Systems

Xing Fu, Khairul Kabir, and Xiaorui Wang
University of Tennessee, Knoxville, TN 37996
{xfu1, kkabir, xwang}@eecs.utk.edu

***Abstract*— Multi-core processors are anticipated to become a major development platform for real-time systems. However, existing power management algorithms are not designed to sufficiently utilize the features available in many multi-core processors, such as shared L2 caches and per-core DVFS, to effectively minimize processor energy consumption while providing real-time guarantees. In this paper, we propose a two-level utilization control solution for energy efficiency in multi-core real-time systems. At the core level, our solution addresses two optimization objectives: controlling the CPU utilization of each core to its desired schedulable bound and minimizing the core energy consumption by adopting per-core DVFS and dynamic L2 cache partitioning to adapt both the CPU frequency-dependent and independent portions of the task execution times of the core. Since traditional control theory cannot handle multiple optimization objectives, a novel utilization controller is designed based on advanced multi-objective model predictive control theory. At the processor level, a cache demand arbitrator is proposed to coordinate the cache size demand from each core and conduct dynamic cache resizing to minimize the leakage power consumption of the shared L2 caches. The energy and time overheads of the proposed control solution are analyzed and addressed in the experiments with well-known benchmarks. Our extensive results demonstrate that our solution outperforms two state-of-the-art power management algorithms that do not consider L2 caches or per-core DVFS, by having more accurate utilization control and less energy consumption.**

I. Introduction

In recent years, more and more real-time systems are developed based on multi-core processors. However, despite a significant amount of existing work on power management for traditional multi-processor real-time systems, existing power management algorithms are not designed to sufficiently utilize the new features available in many multi-core processors, such as shared L2 caches and per-core DVFS (Dynamic Voltage and Frequency Scaling), to effectively minimize processor energy consumption while providing real-time guarantees. For example, most current power/energy management algorithms assume that all the cores of a processor can only have a uniform DVFS level while per-core DVFS is already available (*e.g.*, AMD's Independent Dynamic Core Technology) to allow better power/energy efficiency. Intel's new 48-core processor also features per-tile DVFS with two cores within each tile. In addition, the current algorithms are not designed to dynamically partition the shared L2 caches among the different cores for better real-time performance and to conduct dynamic cache resizing to place rarely accessed cache units into low-power modes for minimized cache leakage power consumption. Therefore, novel power management algorithms are needed to utilize the shared L2 caches and per-core DVFS for maximized energy savings.

In the meantime, many of today's real-time applications commonly execute in open and *unpredictable* environments in which both workloads and system conditions are unknown and may vary significantly at runtime. Traditional task scheduling approaches cannot be directly adopted to provide real-time guarantees for those systems as they rely on Worst-Case Execution Times (WCETs) for schedulability analysis in an open-loop manner. As a result, these approaches may violate the desired timing constraints or severely under-utilize the system when task execution times are highly unpredictable. Likewise, existing power management solutions also focus heavily on open-loop solutions such as static speed scheduling and offline DVFS configurations. For example, many existing speed scheduling algorithms optimize energy/power based on WCETs and thus may fail the optimization goal at runtime because the actual execution times can be much smaller than the WCETs. Therefore, self-adaptive solutions must be developed to dynamically achieve both real-time guarantees and energy efficiency for multi-core real-time systems.

In recent years, feedback-based *CPU utilization control* [19] has been shown to be an effective method of providing soft real-time guarantees by adapting to workload variations based on dynamic feedback. The goal of utilization control is to enforce appropriate schedulable utilization bounds (*e.g.*, Liu and Layland bound) on all the processors in a real-time system, despite significant uncertainties in system workloads. As a result, utilization control can meet all the real-time deadlines without accurate knowledge of the workload such as task execution times. Power-aware utilization control has also been recently proposed to achieve both real-time guarantees and reduced power consumption [29]. However, the existing research on power-aware utilization control primarily relies on DVFS by assuming that the task execution times can be adapted linearly with the CPU frequency. While this assumption is valid for real-time tasks that are computation intensive, memory-intensive tasks can have approximately 75% of their instructions that are load or store [23], [11]. Consequently, when a processor core is running memory-intensive tasks and the CPU frequency is set to the highest level, the utilization can still be above the desired schedulable bound, resulting in undesired deadline misses. In this case, the cache size

This work was supported, in part, by NSF under CAREER Award CNS-0845390 and Grants CNS-0915959 and CCF-1017336, and by ONR under a 2011 Young Investigator Award and Grant N00014-09-1-0750.

partitioned to the core can be increased to reduce the cache miss rate and cache access latency due to reduced main memory access delay. As a result, the CPU utilization can be lowered for better real-time performance. Similarly, if the utilization is lower than the bound, even when the frequency is already throttled to the lowest level, the active cache size can be reduced and rarely accessed cache units can be put into low-power modes to minimize cache leakage power.

In this paper, we propose a two-level utilization control solution for energy efficiency in multi-core real-time systems. At the core level, our solution utilizes both per-core DVFS and dynamic L2 cache partitioning to address two (often conflicting) optimization objectives: controlling the CPU utilization of each core to its desired schedulable bound and minimizing the core energy consumption. Since the utilization contributed by a periodic real-time task is determined by both its CPU frequency-dependent and frequency-independent execution times [5], per-core DVFS and cache partitioning can be used to adapt the frequency-dependent and independent portions, respectively. A key challenge in our design is that traditional control theory, such as PID (Proportional-Integral-Derivative) and MPC (Model Predictive Control), cannot effectively handle multiple optimization objectives. Therefore, we propose a novel utilization controller, based on advanced Multi-Objective MPC control theory [20][3], to achieve both optimization objectives. At the processor level, a cache demand arbitrator is proposed to coordinate the cache size demand from each core and conduct dynamic cache resizing to minimize the leakage power consumption of the shared L2 caches.

Specifically, this paper makes four major contributions:

- We derive an analytic model that captures the system dynamics of the new cache-aware multi-core utilization control problem.
- We propose a two-level utilization control solution for energy efficiency that includes a core-level utilization controller and a processor-level cache demand arbitrator.
- We apply the recent advance in control theory, Multi-Objective MPC (MOMPC) theory, to design the utilization controller for achieving the two (often conflicting) optimization objectives.
- We present extensive experimental results (using the well-known *Mibench* [11] benchmarks) to demonstrate that our solution outperforms two state-of-the-art power management algorithms that do not consider L2 caches or per-core DVFS by having more accurate utilization control and less energy consumption.

The remainder of this paper is organized as follows. We formulate the new cache-aware multi-core utilization control problem in Section II. Section III presents the system model and control architecture. Section IV provides the detailed design and analysis of the MOMPC controller. Section V introduces our simulation environment. Section VI presents our experimental results. Section VII reviews the related work. Finally, Section VIII summarizes the paper.

II. Problem Formulation

In this section, we formulate the cache-aware utilization control problem for multi-core real-time systems.

A. Task Model

A multi-core real-time system is comprised of n cores $\{C_i|1 \leq i \leq n\}$ and m_i periodic tasks $\{T_{ij}|1 \leq j \leq m_i\}$ executing on C_i. Each task T_{ij} has a *soft* deadline related to its period. We use partitioned scheduling to assign tasks to the cores in a multi-core processor. The tasks on each core are scheduled with rate-monotonic scheduling (RMS). Partitioning-based RMS transforms the multi-core real-time scheduling problem into the uniprocessor scheduling problem. A well-known approach to meeting task deadlines on a core is to keep the core utilization below its schedulable utilization bound (e.g., Liu and Layland bound for RMS) [18]. A more precise schedulability test (e.g., the hyperbolic bound [4]) can be used to improve schedulability. Previous studies [1] also show that the Liu and Layland bound can be replaced with the corresponding schedulable utilization bound to ensure timeliness for systems with aperiodic tasks.

Our task model has three important properties. First, while each task T_{ij} has an *estimated* execution time c_{ij} available at design time, its *actual* execution time may be different from the estimation and vary at runtime for several reasons, such as workload variations and intra-core and inter-core interference. Modeling such uncertainties is important to real-time systems operating in unpredictable environments. Second, the L2 caches can be partitioned among the cores. The partition size for each core C_i may be dynamically adjusted. Implementation issues related to cache partitioning and resizing are discussed in Section V. Third, the CPU frequency of each core C_i may be dynamically adjusted within a range $[F_{min,i}, F_{max,i}]$ as many of today's processors support the DVFS technique. Processors that do not support DVFS can use clock modulation to change the frequency at runtime.

B. Problem Formulation

Cache-aware power management for multi-core real-time systems can be formulated as a dynamic constrained optimization problem. We first introduce some notation. T_s, the control period, is selected so that multiple instances of each task are released during a control period. $u_i(k)$ 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 time interval $[(k-1)T_s, kT_s)$. $u_i(k)$ is calculated according to the statistics generated by the operating systems. U_i is the desired utilization set point of C_i. $p(k)$ is the power consumption of the processor and related to both the core frequencies and active L2 cache size. $E(k)$ is the energy consumption of the processor in the k^{th} control period. Since the core frequencies, active L2 cache size, and workload of the processor are all not changed during a control period, $p(k)$ can be approximated as a constant within each control period. Consequently, $E(k) = p(k)T_s$. We assume that the processor has homogeneous cores with two levels of caches and the L2 caches are shared among the cores since mainstream multi-core processors adopt this architecture. We

also assume that the processor supports per-core DVFS as per-core DVFS leads to a better processor energy efficiency than a chip-wide DVFS [13]. We further assume the cache can be partitioned among tasks. The details of dynamic cache partitioning is beyond the scope of this paper because various ways (e.g., software or hardware) have already been designed to implement cache partitioning among tasks. Examples can be found in [17][24][14][10][8]. $s_i(k)$ is the L2 cache partition size of core C_i. $f_i(k)$ is the relative core frequency (*i.e.*, the core frequency relative to the highest level $F_{max,i}$) of core C_i.

Given a utilization set-point vector, $\mathbf{U} = [U_1 \dots U_n]^T$, a frequency constraint $[F_{min,i}, F_{max,i}]$ for each core C_i, and the total L2 cache size S for the processor, the control goal at the k^{th} sampling point (time kT_s) is to dynamically choose the cache partition size $\{s_i(k)|1 \le i \le n\}$ and core frequency $\{f_i(k)|1 \le i \le n\}$ to minimize the difference between U_i and $u_i(k)$ for all the cores and to minimize the energy consumption $E(k)$ for the processor.

$$\min_{s_i(k)|1\le i\le n, f_i(k)|1\le i\le n} \sum_{i=1}^{n} [U_i - u_i(k)]^2 \tag{1}$$

$$\min_{s_i(k)|1\le i\le n, f_i(k)|1\le i\le n} E(k) \tag{2}$$

subject to

$$F_{min,i} \le f_i(k) \le F_{max,i} \qquad (1 \le i \le n) \tag{3}$$

$$\sum_{i=1}^{n} s_i(k) \le S \tag{4}$$

Note that the objective (2) is actually equivalent to the minimization of power consumption because the power consumption during a control period can be approximated as a constant and thus $E(k) = p(k)T_s$. Constraint (3) guarantees that the CPU frequency of each core remains within its acceptable range. The frequency range depends on specific processors. The above formulation can be extended to add equality constraints among cores that have the same frequency (and voltage). Constraint (4) ensures that the summed size of all the cache partitions does not exceed the total available cache size on the processor. For each core, the optimization formulation minimizes the difference between the core utilization and corresponding set point by manipulating both partition size and core frequency while satisfying the constraints. Control goal (1) actually may conflict with control goal (2) because core frequencies throttled to the lowest levels and cache lines turned off are desired to minimize the total power consumption $p(k)$. In that case, memory accesses would be very slow because most accesses will face cache misses and non-memory access instructions would be executed at the lowest speed. Consequently, the task execution times would be too long and core utilizations might exceed set points, leading to deadline misses. Therefore, the two conflicting goals require resolution with advanced control and optimization techniques.

III. Cache-Aware Utilization Control

In this section, we model the cache-aware utilization control problem for energy efficiency in multi-core real-time system and present our two-level control architecture.

A. System Modeling

Following a control-theoretic methodology, we establish a dynamic model that characterizes the relationship between the controlled variable $u_i(k)$ and manipulated variables $s_i(k)$, and $f_i(k)$ in the k^{th} control period, by system identification. First, we model the relationship between $c_{ij}(k)$, the execution time of task T_{ij} running on core C_i, and the two manipulated variables, $f_i(k)$ and $s_i(k)$. According to previous research [5], $c_{ij}(k)$ normally consists of frequency-dependent and frequency-independent portions

$$c_{ij}(k) = \frac{n_{ij}}{f_i(k)} + m_{ij}(k) \tag{5}$$

where $\frac{n_{ij}}{f_i(k)}$ is the frequency-dependent portion and $m_{ij}(k)$ is the frequency-independent portion of T_{ij}'s execution time. The former scales with the core frequency but the latter does not because some instructions deal with memory or other I/O devices and their access speeds do not depend on core frequency. For processors whose FSB (front-side bus) speed varies with DVFS, memory accesses delay can be modeled as the frequency-dependent portion of the task execution time. We assume that the data and program of real-time tasks are loaded into main memory. Disk or I/O device accesses are not required during the execution. The assumption is valid for the majority of embedded real-time systems as the memory footprints of those applications are typically small. Intuitively, $m_{ij}(k)$ is related to the cache size reserved for T_{ij} because of the strong correlation between the cache size of an application and the number of cache misses it has. According to [8], the relationship between $m_{ij}(k)$ and $s_{ij}(k)$, the cache size allocated to T_{ij} on C_i, is modeled as

$$m_{ij}(k) = \begin{cases} A_{ij}\mathrm{s}_{ij}(k) + B_{ij} & 0 \le \mathrm{s}_{ij}(k) \le W_{ij} \\ \text{Constant} & \mathrm{s}_{ij}(k) > W_{ij} \end{cases} \tag{6}$$

where W_{ij} is the working set size (WSS) of task T_{ij}. A_{ij} and B_{ij} are task-specific parameters. All the parameters can be estimated using existing task profiling techniques. Example parameters for the benchmarks used in our experiments are listed in Table I in Section V. When $\mathrm{s}_{ij}(k)$ is smaller than the WSS W_{ij}, increasing the cache size of a task may lead to a reduced execution time [8]. When the allocated cache size is greater than the WSS, allocating additional cache to a task does not further decrease its cache miss rate. Although model (6) is an approximation of the real system, our experiments show that the linear relationship is sufficiently accurate for the benchmarks. When a workload is different from the benchmarks, it can be proved that the proposed solution still achieves the control goal if the execution time varies within a specific range.

For preemptive real-time task systems, we can establish the following relationship between the total frequency-independent execution time of all the tasks on core C_i and the total cache size $s_i(k)$ assigned to C_i

$$m_i(k) = \begin{cases} \sum_j A'_{ij} s_i(k) + \sum_j B_{ij} & 0 \le s_i(k) \le W_i \\ \text{Constant} & s_i(k) > W_i \end{cases} \tag{7}$$

where $A'_{ij} = \frac{A_{ij}s_{ij}(k)}{s_i(k)}$ and $W_i = \sum_j W_{ij}$. (7) is derived by a sum of (6) across all the tasks on core C_i. We assume that each task has its own cache partition. Note that we do not need to reserve caches for every task on each core and divide $s_i(k)$ proportionally. In that case, the overhead that occurs because the cache content can be invalidated by preempting tasks is taken into consideration. It depends on the maximum number of times the task is preempted and the cache size the task is using. So, we use the WCET model from [8] and derive a model similar to (7).

In multi-core systems, tasks on different cores may compete and interfere with each other for shared resource (e.g., shared bus or caches) access. To avoid these interferences, we adopt the cache partitioning method proposed in multiple studies (*e.g.*, [17]). The multi-core cache architecture in [17] simplifies the WCET analysis of a real-time multi-core system. Without cache partitioning, unpredictable inter-core interferences may occur and invalidate model (7). Based on this architecture, a multi-core processor with shared L2 caches can be regarded as a multiprocessor system with each processor having adjustable private L2 caches. Considering (5), (6), and (7), we derive the following model for our system

$$b_i(k) = \frac{\sum_j n_{ij}r_{ij}}{f_i(k)} + \sum_j A'_{ij}r_{ij}s_i(k) + \sum_j B_{ij}r_{ij} \quad (8)$$

where $b_i(k)$ is the estimated utilization of core C_i and r_{ij} is the task rate of T_{ij} running on that core. An important observation is that system model (8) needs to be transformed as $b_i(k)$ to be inversely related to the core frequency $f_i(k)$. From system model (8), the estimated change of utilization, $\Delta b_i(k)$, for core C_i is modeled as

$$\Delta b_i(k) = d_i(k)\sum_j n_{ij}r_{ij} + \Delta s_i(k)\sum_j A'_{ij}r_{ij} \quad (9)$$

where $d_i(k) = \frac{1}{f_i(k)} - \frac{1}{f_i(k-1)}$ and $\Delta s_i(k) = s_i(k) - s_i(k-1)$. Now $\Delta b_i(k)$ is a linear function of $d_i(k)$ and $\Delta s_i(k)$, which allows us to use $d_i(k)$ as the manipulated variable instead of using $f_i(k)$ directly. Note that $\Delta b_i(k)$ depends on the estimated values of n_{ij} and A'_{ij}. Their actual values may be different from the estimations due to workload variations. A major contribution of our work is to propose a control solution to handle this uncertainty.

The system model (9) represents a Multi-Input-Single-Output (MISO) system because it has two manipulated variables, $d_i(k)$ and $\Delta s_i(k)$, and one controlled variable. Two manipulated variables can provide extra flexibility for controlling both CPU-intensive and memory-bound tasks when compared with controlling the same tasks with only one manipulated variable. The additional input variable has a significant implication on the control solution design. We can achieve a certain output with an infinite number of combinations of these two inputs, but not all of them can satisfy the utilization control and power optimization goals. Therefore, we need to determine which combination to use to fulfill our goals. The details are discussed in Section IV.

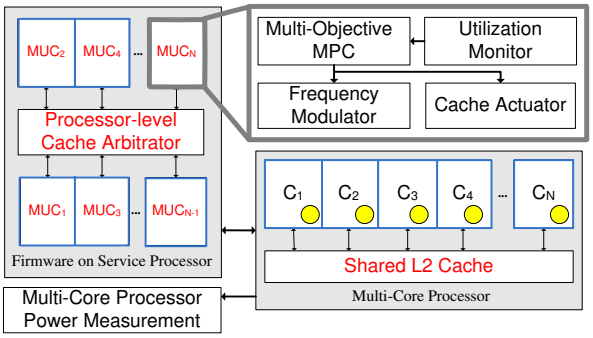


Fig. 1. Two-level utilization control architecture.

From the system perspective, in multi-core environments that allow both DVFS and cache partitioning/resizing, relying solely on one adaptation strategy may unnecessarily reduce the system's adaptation capability. Adapting one of them can only adjust either the frequency-dependent or independent portion of the task execution time within a range, but not both. Therefore, a novel control architecture needs to be designed for utilization control and power management in multi-core real-time systems by utilizing both adaptation strategies.

B. Control Architecture

In this paper, we propose a novel two-level utilization control and power management architecture. As shown in Figure 1, our control architecture features a core-level utilization controller and processor-level cache demand arbitrator. As described in Section II, constraint (4) enforces that the summation of $s_i(k)$ should not exceed the total processor cache size. Therefore, if the partition size of a core is increased, the cache sizes of other cores may need to be reduced. Moreover, the utilization of a core is related to its cache partition size according to system model (9). The coupling between the cache size and core frequency for utilization control raises new design challenges. Instead of designing a single processor-level utilization controller, we adopt the two-level utilization control architecture based on the following considerations.

First, a processor-level utilization controller may not scale well in future many-core systems (*i.e.*, systems with tens or hundreds of cores), because the number of variables in the system model of the processor-level utilization controller increases proportionally with the number of cores. As a result, the computational complexity of the controller can increase significantly and thus be too expensive to control real-time systems. In addition, whenever the number of cores changes, the system model changes and the controller needs to be redesigned. Core-level controllers have better scalability because the number of controlled and manipulated variables does not increase with the number of cores. However, as a core-level controller determines its own cache partition size locally and is unaware of other core's cache demands, it can not guarantee constraint (4). Therefore, a processor-level cache demand arbitrator is needed to enforce the constraint by

assigning a cache quota $s_{quota,i}$ to each core. The core-level local controller maintains its cache partition size below the cache quota $s_{quota,i}$ assigned by the arbitrator.

Second, as the feature size is shrinking to the nanometer scale, leakage power becomes the dominant portion of the total power consumption of the entire processor. The leakage power of a processor contains leakage power for both the cores and caches. In this paper, we reduce the cache leakage power by resizing L2 caches at runtime to turn off unused portions. Our solution can also be integrated with existing task migration policies to migrate real-time tasks among the cores and turn off idle cores to reduce the core leakage power. Note that task migration is complementary to our solution and that detailed integration is beyond the scope of this paper. Task migration for power efficiency can be better supported with a core-level utilization controller than with a processor-level utilization controller. The reason is that the number of active cores may change at runtime and the system model of the processor-level MPC controller needs to be rebuilt whenever the number of the active cores changes. This may incur a large overhead to the system.

Our two-level utilization control architecture works as follows. First, the processor-level cache demand arbitrator dynamically calculates a cache quota for every core based on the real-time workloads running on them. It monitors the task arrival, termination, and migration events, periodically, to collect the cache demand of every core. The core-level utilization controller uses this cache quota to enforce the constraint (4). Second, each core-level controller controls the utilization of the corresponding core by scaling its frequency and resizing its cache partition. It is a MISO controller that adopts advanced MPC theory to serve this multi-objectivity: utilization control and power optimization. The core-level controller executes the following steps at the end of every control period: (1) It collects the core utilization from the utilization monitor on core C_i; (2) The controller then computes a new core frequency $f_i(k)$ and a new cache partition size $s_i(k)$, then sends the values to the frequency modulator and cache actuator on C_i, respectively; and (3) The frequency modulator and cache actuator change the core frequency and cache partition size accordingly. In a real system, similar to the power management unit implemented in POWER7, our control architecture can be implemented in service processor firmware that manages the controlled multi-core processor.

IV. MOMPC Controller Design

In this section, we present the formulation of the MOMPC controller and discuss the controller design in detail.

A. MOMPC Control

Based on system model (9), a novel MISO controller needs to be designed to enforce the utilization set points on all the cores and minimize power consumption of the processor simultaneously. Traditional MPC control theory applied in earlier studies on feedback control real-time scheduling (*e.g.*, [19]) is not suitable for the problem we formulate in Section II. The reason is that traditional MPC theory can not handle multiple control goals like the two we have in our problem. To solve our control problem, we adopt a recent advance in control theory, Multi-Objective Model Predictive Control (MOMPC) [20], which is being actively studied in the control community [3]. One of the advantages of MOMPC is its capability of dealing with multi-objective MIMO control problems with constraints on the plant and actuators. This characteristic makes MOMPC suitable for our problem.

The basic idea of MOMPC control is to solve a hierarchy of optimization problems. Specifically, multiple objectives are ranked according to their priorities since they may conflict with each other and cannot be met simultaneously. In MOMPC control, the most important objective is solved first. The solution is then used to impose equality constraints when addressing the second optimization objective, and so on. Since meeting the real-time constraints is always the first priority in real-time systems, we select objective (1) as our primary control goal and objective (2) as our secondary goal. To meet the two control goals, we have a *primary* optimizer and a *secondary* optimizer. The primary optimizer is essentially a dynamic least square optimizer designed to meet the control goal (1), just like the optimizer in traditional MPC theory. Its control objective is to select a combination of core frequency $f_i(k)$ and cache partition size $s_i(k)$ that achieves *only* the control goal (1). When the system is controlled into the stable state, the secondary optimizer adjusts the core frequency $f_i(k)$ and cache size $s_i(k)$ to achieve the control goal (2), *i.e.*, minimizing the power consumption of the processor. To avoid conflicting with the primary optimizer, the secondary optimizer enforces an equality constraint to adjust the core frequency $f_i(k)$ and cache size $s_i(k)$, without impacting the core utilization $u_i(k)$.

B. Primary Optimizer

Following MOMPC control theory, we first design a controller for the primary optimizer to achieve the control goal (1). The controller employs system model (9) to minimize a cost function with constraints. The cost function to be minimized by the controller for core C_i is

$$V_i(k) = \sum_{l=1}^{P} \left\| u_i(k+l-1|k) - ref_i(k+l-1|k) \right\|^2 + \left\| \mathbf{x_i}(k|k) - \mathbf{x_i}(k-1|k) \right\|^2 \tag{10}$$

subject to:

$$\begin{aligned} F_{\min,i} \le f_i(k) \le F_{\max,i} \\ s_i(k) \le s_{quota,i} \end{aligned} \tag{11}$$

where $\mathbf{x_i}(k) = \begin{bmatrix} d_i(k) \\ \Delta s_i(k) \end{bmatrix}$. P is the prediction horizon used to predict the system behavior over P control periods, $P = 2$ in our system. $ref_i(k+l|k)$ is the reference trajectory along which the utilization vector $u_i(k+l|k)$ should change from the current utilization $u_i(k)$ to the utilization set point U_i. Note that the cache size $s_i(k)$ for C_i is bounded by $s_{quota,i}$ to ensure constraint (4). We can easily transform the above optimization problem into a standard constrained least-square problem that can be solved by the controller

using any standard least square solver. The transformation is not presented due to space limitations, but the detailed steps can be found in [20]. Although the outputs of the primary optimizer are unique, the outputs may not be optimal in terms of energy efficiency. As explained in Section III-A, multiple combinations of core frequencies and cache sizes including the outputs of the primary optimizer can satisfy the utilization set point.

C. Secondary Optimizer

The secondary optimizer uses a power model to achieve the desired control goal (2), *i.e.*, minimizing the power consumption of the processor. The power optimization function that we have designed for our secondary optimizer is

$$p_i(k) = M_i f_i(k)^3 + N_i s_i(k) + L_i \tag{12}$$

subject to

$$\begin{array}{c} F_{\min,i} \leq f_i(k) \leq F_{\max,i} \\ s_i(k) \leq s_{quota,i} \end{array} \tag{13}$$

where M_i, N_i, and L_i are the power model parameters of the processor. The power consumption of the processor includes the power consumed by the cores and caches. The former has a dynamic power component $M_i f_i(k)^3$ that varies with core frequency and a leakage power component L_i, but for the latter, the dynamic power component is negligible when compared with the leakage power component [21]. Thus, the cache power consumption is approximated by $N_i s_i(k)$ which varies with the cache partition size of C_i. The power model parameters in (12) can be a function of processor temperature, which can significantly impact the leakage power.

The secondary optimizer finds a combination of $f_i(k)$ and $s_i(k)$ that minimizes (12) while satisfying the constraints of (13). As previously discussed, the equality constraint is imposed so that adjusting core frequency $f_i(k)$ and cache size $s_i(k)$ does not change core utilization $u_i(k)$ achieved by the primary optimizer. As both $u_i(k)$ and $p_i(k)$ are functions of $f_i(k)$ and $s_i(k)$, we can establish a relationship between them and easily impose the equality constraints. We can transform the above formulation into a standard nonlinear optimization problem with constraints and solve it using any standard solver. The detailed transformation is not presented due to page limitations. In our simulator, we implemented the secondary optimizer based on a Matlab solver (introduced in Section V). The solver can find the optimal solution with a time complexity of $\mathrm{O}(n^3)$.

We configure the control period of the secondary optimizer to be 50 times the control period of the primary optimizer. We have proven that the configuration guarantees the stability of the proposed control solution. The detailed proof is not included due to space limitations.

V. Simulation Environment

Our simulation environment integrates the event-driven EUCON simulator (for real-time task scheduling) used in previous studies [19] and a multi-core cache partitioning system implemented by following the cache implementation of the cycle-accurate SESC simulator [26], which is widely used in computer architecture research. The multi-core processor simulated in our work is an Intel Xeon X5365 Quad Core processor with an 8MB on-die shared L2 cache and 1333 MHz FSB. The processor supports four DVFS levels: 3GHz, 2.67GHz, 2.33GHz, and 2GHz. All the parameters in our power and utilization models are based on the data sheet from Intel or profiling experiments conducted on the real processor. We have validated our models under different DVFS levels and cache partition sizes with the real Intel processor and original SESC simulator, respectively. The validations show that our models are sufficiently accurate (with $R^2 \geq 0.93$) for the well-known *Mibench* [11] benchmark suites designed for embedded systems. We only list the result of the first category benchmarks of MiBench suite among all the six categories because other categories are not designed to test real-time systems. [23]. Table I lists the benchmarks used in our experiments and the corresponding parameters used in model (6). The unit for the working set size (WSS) is the number of cache lines.

TABLE I

System model parameters in (6) for typical benchmarks.

Benchmark	WSS	n_{11}	A_{11}	B_{11}	R^2
basicmath	2026	4.4e+7	-3.8e+7	3.8e+7	0.93
susan	886	4.05e+7	-9673	7.0e+6	0.99
bitcnts	445	7.64e+7	-1.7e+5	9.0e+7	0.99

The simulation environment implements a multi-core real-time system based on the simulated processor and the cache-aware power management and utilization control architecture, which includes the utilization monitors, core frequency modulators, cache partitioning/resizing actuators, and the processor-level cache arbitrator. The periodic tasks on each core are scheduled by RMS. Similar to previous studies based on the EUCON simulator, the multi-objective MPC controllers are implemented in Matlab. Specifically, the primary optimizer of an MOMPC controller is implemented based on the `lsqlin` least squares solver and the secondary optimizer is implemented based on the `fmincon` constrained nonlinear multi-variable optimizer. In each simulation, the simulator first opens a Matlab process and initializes the parameters. At the end of each control period, the simulator collects the utilization of each core from the utilization monitors, and calls the MOMPC controllers in Matlab. The MOMPC controllers compute the control inputs, $f_i(k)$ and $s_i(k)$, and return them to the simulator. The simulator calls the frequency modulators and cache partitioning/resizing actuators to enforce the control inputs. Note that the overhead of the MOMPC controllers is sufficiently small because we adopt the core-level controller design (discussed in Section III). As a result, each MOMPC controller only has one controlled variable and two manipulated variables. Note also that the controllers can be implemented in service processor firmware in a real system and thus its computation and power overheads will not significantly affect the main multi-core processor. An MOMPC controller can also tolerate a considerable communication delay, as long as the delay is short when compared with the

control period [20].

Cache partitioning divides a shared cache into non-overlapping partitions for independent use by real-time tasks. The benefit is that it eliminates the inter-core interferences among real-time tasks caused by the shared cache and thus leads to improved real-time performance [2], because the interferences may introduce difficulties to the estimation of WCETs of real-time tasks. It is well known that the WCET estimation for shared-cache multicores is still an open problem because interferences exist. Given a k-associative cache (not necessarily a fully-associative cache) with l cache sets, the cache can be divided based on associativity or based on cache sets. Associativity-based partitioning assigns a certain number of ways (0 to k) within each cache set to a partition while set-based partitioning assigns a certain number of sets (0 to l) to a partition. The difference of the two approaches is the partitioning granularity. In this work, we design the proposed control solution on set-based partitioning because its granularity is fine-grained ($l >> k$).

Overhead Analysis: Our simulations take into consideration both time and energy overheads of the proposed MOMPC controller, DVFS and cache partitioning. We measure the execution times and energy consumption of both the primary optimizer and the secondary optimizer of the proposed MOMPC controller by running it on the simulated multi-core processor. The time overhead of the primary optimizer is 0.8ms and its energy overhead is approximately 0.088J. The time overhead of the secondary optimizer is 2.2ms and its its energy overhead is approximately 0.242J. Although overheads of the secondary optimizer are higher than those of the primary optimizer, the secondary optimizer is only invoked every 50 control periods. The total time overhead of the proposed MOMPC controller is less than 2% of a control period. Park et al. [25] presents an accurate modeling of the time and energy overheads of DVFS techniques such as Intel's SpeedStep Technology and AMD equivalent PowerNow. The transition time is between 15.2 μs to 82.6 μs and its energy overhead is from 0.1 mJ to 0.52 mJ. Therefore, the time overhead of DVFS is less than 0.6% of the control period. To implement the cache partitioning in a chip, additional circuits have to be added which will consume additional energy compared with the processors without cache partitioning support. Studies on computer architeture [17][14] have shown that the time overhead is 2% on average. The area for the circuits implementating some cache partitioning technique is only 1.5% of the total area of caches. Thus, the energy overhead of the cache partitioning is estimated to be 1.5% of the energy consumption of caches. In our simulations, we deduct all the estimated energy overheads related to the proposed control solution from the energy results.

VI. Experimental Results

In this section, we first introduce two state-of-the-art baselines. We then evaluate our proposed control architecture using the Mibench benchmarks and compare it with the baselines.

A. Baselines

Our first baseline, referred to as *Dynamic repartitioning* [27], is a typical energy-efficient scheduling algorithm for real-time tasks on a multi-core processor without considering the frequency-independent component of task execution time and cache power consumption. To achieve a low power consumption, Dynamic repartitioning balances the dynamic utilization of all cores by migrating tasks among the cores. It calls a repartitioning function whenever a task is completed or a new task period starts. The function migrates a task T_m that lowers the chip-wide frequency after migration, from the core with the highest dynamic utilization, C_{max}, to the core with the lowest dynamic utilization, C_{min}. The migration process continues until the chip-wide frequency level cannot be lowered further by task migration. The key differences between Dynamic repartitioning and our solution are that 1) Dynamic repartitioning assumes the task execution time scales inversely linearly with the core frequency and 2) all the cores in a processor are assumed to have a uniform DVFS level.

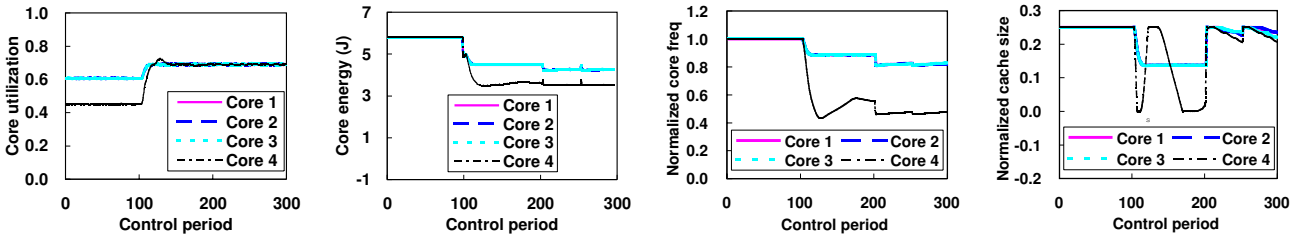
The second baseline, referred to as *DVFS-Only*, is the frequency scaling loop proposed in [29]. DVFS-Only represents existing utilization control mechanisms that assume the task execution time scales only with the CPU frequency and applies DVFS for utilization control and power management.

We show that our proposed solution, which manipulates both frequency and cache size, outperforms both baselines by consuming less power consumption.

B. Cache-Aware Utilization Control

In this experiment, we first evaluate the performance of our MOMPC controller. We adopt two different task sets to conduct our experiments on the simulated quad-core processor. The first task set includes two periodic tasks running *basicmath* benchmarks with a total utilization of 0.6, while the second task set contains three periodic tasks running a mix of *basicmath* and *bitcnts* benchmarks with a total utilization of 0.45. The workloads for the first three cores are identical and they execute the first task set. The workload for core 4 is different and it executes the second task set. The task period of basicmath is 0.08 seconds while the task period of bitcnts is 0.16 seconds. We initially assign an even cache quota to each core. We also conduct a set of experiments to examine randomly generated workloads.

In our experiment, we activate our MOMPC controllers on all the cores at time 100 (*i.e.*, the 100^{th} control period) and enable the secondary optimizers at time 200. Figure 2(a) shows that the utilizations of all cores are controlled accurately to their RMS bounds (*e.g.*, 0.69) after the MOMPC controllers are activated. As a result, no deadline miss is observed. Figure 2(b) shows that the energy consumption in every control period. The specific value of the control period is 0.16 seconds. The energy consumption are reduced from time 100 to 200. After the secondary optimizers are enabled on all the cores at time 200, the energy consumption is minimized: from 6J to 4J for cores 1-3 and from 6J to 3.5J for core 4. The small spikes in the energy consumption at the 200th and 250th control periods are caused by the secondary optimizer. Figures 2(c) and 2(d) detail the behavior of the MOMPC controllers by plotting the frequencies and cache partition sizes of the cores. From time 100 to 200, the MOMPC controllers, without the

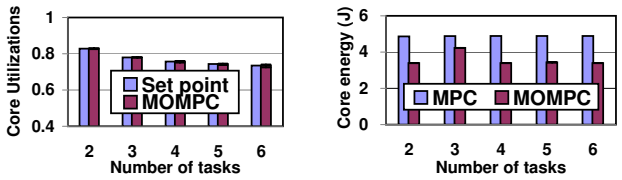


(a) Core utilization (b) Core energy consumption (c) Core frequency (d) Core cache size

Fig. 2. A typical run of the proposed cache-aware control solution. The MOMPC controllers (primary optimizers) are activated at time 100 to control utilizations and the secondary optimizers are enabled at time 200 for energy optimization.

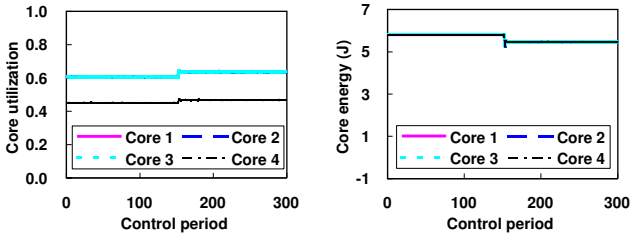
secondary optimizers, does not reduce the frequencies to the minimum level. As a result, the processor energy consumption is not minimized. At time 200, the secondary optimizers are enabled to achieve energy optimization by throttling the core frequencies. As a result, the cache partition sizes are increased for all the cores and overall energy consumption is reduced without affecting the core utilizations. This experiment clearly demonstrates that the MOMPC controller can achieve better energy efficiency than a traditional MPC controller that does not contain the secondary optimizer.

To test the robustness of the proposed MOMPC controller, we conduct a set of experiments with different randomly generated workloads. For each workload, the number of tasks on each core is increased from 2 to 6 (*i.e.*, 8 to 24 tasks in total). Figure 3(a) plots the average CPU utilizations of all the cores after the controllers enter the steady state. Our MOMPC controllers successfully achieve the desired utilization set points with zero steady state errors for all the workloads. Figure 3(b) shows that the MOMPC controllers achieve more energy savings than the MPC controllers.



(a) Core utilization (b) Core energy consumption

Fig. 3. The proposed cache-aware solution (i.e.,MOMPC) controls core utilization to desired set points while saving more energy than MPC.



(a) Core utilization (b) Core energy consumption

Fig. 4. A typical run of the baseline Dynamic repartitioning (activated at time 150).

C. Comparison with Dynamic Repartitioning

In this experiment, we compare the proposed solution with the first baseline, Dynamic repartitioning. To have a fair comparison, we adopt the same workload used in Section VI-B. Figure 4(a) shows that after Dynamic repartitioning activates at time 150, the utilizations of all the cores increase only slightly. None of the cores achieve the desired utilization set points (*e.g.*, 0.69). The reason is that Dynamic repartitioning assumes that the execution times of tasks are inversely proportional to the core frequencies, without considering the frequency-independent execution times. In the workload we adopted, the frequency-independent execution times (about 2.83e+7 CPU cycles) comparable to the frequency-dependent execution times (about 4.4e+7 CPU cycles). As a result, Dynamic repartitioning fails to control utilizations accurately, which can lead to power inefficiency, as shown later. Another fundamental assumption of Dynamic repartitioning is chip-wide DVFS, which holds true for certain multi-core processors. However, as microelectronic technologies advance, per-core DVFS has been implemented and is expected to become the main-stream configuration. Since the workload on each core is not perfectly balanced, the cores cannot achieve their utilization set points simultaneously with chip-wide DVFS.

Figure 4(b) shows the energy consumption of each core. Since Dynamic repartitioning reduces the frequencies of all the cores from the highest level to the same level (with chip-wide DVFS) and does not manage the energy consumption of the shared L2 caches, the energy savings of each core is approximately identical. Even though no deadline is violated by Dynamic repartitioning, the energy consumption of each core is only slightly reduced from 6J to 5.5J, which leads to unnecessarily more energy consumption as the proposed cache-aware control can reduce energy significantly. The first reason is that Dynamic repartitioning can not control utilizations accurately. Since the proposed cache-aware control considers frequency-independent execution times, both core frequencies and cache partitions can be adjusted to achieve accurate utilization controls which translate to additional energy savings. The second reason is that Dynamic repartitioning does not take advantage of per-core DVFS, which is proven to be more energy efficient than chip-wide DVFS [13]. This experiment demonstrates that the cache-aware control solution outperforms Dynamic repartitioning in terms of energy efficiency.

To stress test the energy efficiency of the proposed solution and Dynamic repartitioning, we conduct a series of experiments with different workloads. For each workload, the frequency-dependent execution times of the tasks are changed by tuning a parameter called the *execution-time factor*

(etf). The etf is the ratio of the frequency-dependent portion to the frequency-independent portion in the task execution times of the workload. The number of loop iterations in the computation-intensive benchmarks (e.g., *susan* or *bitcnts*) can be adapted to vary the frequency-dependent execution times of the mixed benchmark consisting of *basicmath* and *bitcnts*. Figure 6 shows that the energy consumption of Dynamic repartitioning is always higher than that of the proposed cache-aware solution in a wide range of etfs. As the etf increases, the frequency-independent portion becomes relatively smaller, thus the gap between the two solutions narrows. The reason is that when the the frequency-independent portion becomes relatively smaller, the advantage of dynamic cache resizing used by the proposed solution for reduced leakage power becomes smaller. As a result, both solutions primarily rely on DVFS for power efficiency.

D. Comparison with DVFS-Only

In this experiment, we compare the proposed solution with the second baseline: DVFS-Only. We activate the solutions at time 100. The workload on each core is configured to be identical and includes three periodic tasks. We simulate the typical scenario of a real-time system with uncertain execution times by increasing the frequency-dependent execution times of the tasks on all the cores by 100% at time 250. We compare the energy efficiency of the proposed solution and DVFS-Only in such a scenario.

Figure 5(a) shows that under the proposed solution, after time 100, the utilizations of all the cores are controlled to the set points (e.g., 0.69). Due to the workload variation at time 250, the utilizations increase significantly. The proposed solution successfully controls the utilizations back to the set points. The deadline miss rate is 0.5% since the utilization bound approximately at time 250 is violated, creating deadline misses. Figure 5(b) shows that before the proposed solution activates at time 100, the core energy consumption is high because the core frequencies are initially set to the highest levels and the L2 caches are all turned on. From time 100 to 250, the energy consumption is reduced significantly by the proposed solution. After time 250, both the core frequencies and cache sizes are increased due to the workload increase, resulting in an increased energy consumption.

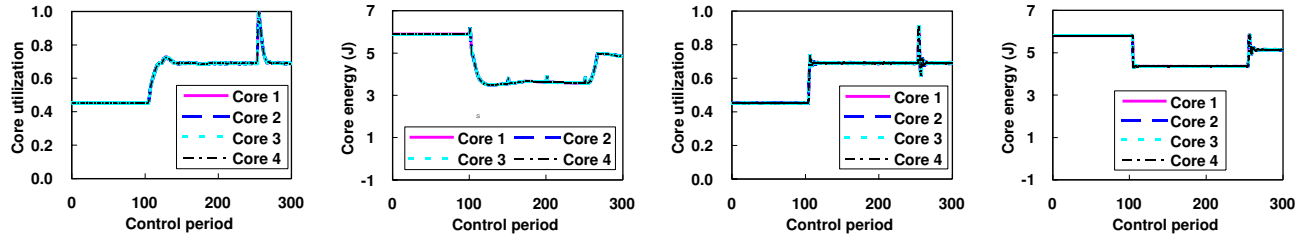
Figure 5(c) shows a typical run of DVFS-Only in the same scenario under the same workload. Note that although DVFS-Only also assumes the execution times of tasks are inversely proportional to the core frequencies as Dynamic repartitioning, DVFS-Only can control the utilizations to the set points accurately because DVFS-Only relies on the feedback of the measured utilizations. When compared with the proposed solution, the deadline miss ratio of DVFS-Only is zero because the peak utilization of the proposed solution is 1 while the peak utilization of DVFS-Only is only 0.9. Figure 5(d) shows that from time 100 to 250, the energy consumption is approximately 4.3J, which is much higher than 3.7J, the power consumption of the proposed solution (shown in Figure 5(b)). The reason is that DVFS-Only does not turn off the caches for energy savings. Thus, the energy consumption cannot be reduced significantly by only throttling DVFS. After time 250, the energy consumption is approximately 5.1J while the power consumption of the proposed solution is about 4.3J. As the frequency-dependent portion in the execution times increases, the gap of the energy consumption of the two solutions narrows. On average, DVFS-Only consumes 20% more energy per core than the proposed solution. This experiment demonstrates that the proposed solution is more energy efficient than DVFS-Only under workload variations.

To test the impact of the parameters in the power model on energy efficiency, we define the *power ratio* of a core to be the ratio of the dynamic power consumption when the core frequency is the maximum level to the cache power consumption of the core when all the caches are turned on. We use the same scenario to increase the workload at time 250. Figures 7 and 8 show the energy consumption of the two solutions before and after time 250 (workload increase), respectively. Figure 7 shows that when the power ratio is lower, which means the percentage of leakage power consumption is higher in the total power consumption, the gap between the proposed solution and DVFS-Only widens because DVFS-Only can only adjust DVFS to manage power consumption. The difference between the proposed solution and DVFS-Only in Figure 7 is smaller than that in Figure 8. The reason is that when the frequency-independent execution times become relatively smaller, the advantage of the proposed cache-aware solution to dynamically resize caches for reduced leakage power becomes smaller.

VII. Related Work

In recent years, scheduling for multi-core real-time systems has received much attention. Many multiprocessor scheduling algorithms (*e.g.*, [9], [22]) can be applied to multi-core processors. Bini et al. [6] proposed two abstractions to facilitate multi-core adoption for real-time systems and the corresponding schedulability analysis. Nelis et al. [22] studied slack reclamation schemes to reduce the power of a multi-core real-time system. Block et al. [7] proposed an adaptive framework based on feedback which controls each task instead of the utilization of the task system. Seo et al. [27] studied energy efficient multi-core real-time scheduling using a chip-wide DVFS. However, all these studies do not explicitly consider the impact of shared L2 caches.

Several cache-aware multi-core real-time scheduling algorithms have been recently proposed. Anderson's group proposed various open-loop cache-aware global scheduling algorithms for multi-core real-time systems (*e.g.*, [2]). Lakshmanan et al. [15] studied partitioned fixed-priority preemptive scheduling. Bui et al. [8] optimized the impact of cache partitioning on a multi-core real-time system. Guan et al. [10] also studied cache-aware scheduling. Yan et al. [30], Li et al. [16] and Hardy et al. [12] analyzed the impact of a shared L2 instruction cache on WCET estimation for shared L2 cache multi-core systems. Paolieri et al. [24] used L2 cache partitioning to solve the multi-core WCET problem. Suhendra et al. [28] proposed a similar cache partitioning and locking approach. All the aforementioned studies are different from



(a) Core utilization (Cache-aware) (b) Core energy (Cache-aware) (c) Core utilization (DVFS-Only) (d) Core energy (DVFS-Only)

Fig. 5. Typical runs of cache-aware control and DVFS-Only under workload variations.

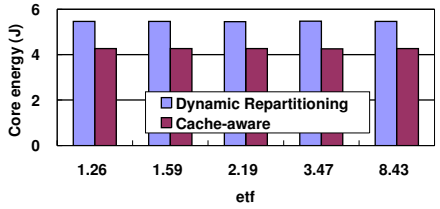


Fig. 6. Comparison with Dynamic repartitioning when etf varies.

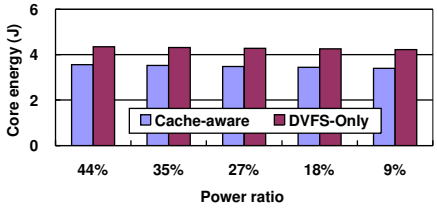


Fig. 7. Comparison with DVFS-Only when power ratio varies (before workload increases).

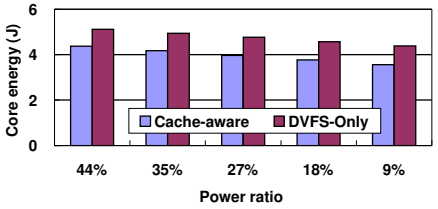


Fig. 8. Comparison with DVFS-Only when power ratio varies (after workload increases).

ours because they do not address the power consumption of a shared L2 cache.

VIII. Conclusions

In this paper, we have presented a two-level utilization control solution for energy efficiency in multi-core real-time systems. At the core level, our solution addresses two optimization objectives: controlling the CPU utilization of each core to its desired schedulable bound and minimizing the core power consumption by adopting per-core DVFS and dynamic L2 cache partitioning to adapt both the CPU frequency-dependent and independent portions of the task execution times of the core. Since traditional control theory cannot handle multiple optimization objectives, a novel utilization controller is designed based on advanced MOMPC theory. At the processor level, a cache demand arbitrator is proposed to coordinate the cache size demand from each core and conduct dynamic cache resizing to minimize the leakage power consumption of the shared L2 caches. The energy and time overheads of the proposed control solution are analyzed and demonstrated to be sufficiently small. Extensive experiments using the Mibench benchmarks show that our solution outperforms two state-of-the-art power management algorithms that do not consider L2 caches or per-core DVFS by having more accurate utilization control and less energy consumption.

References

[1] T. F. Abdelzaher, V. Sharma, and C. Lu, "A utilization bound for aperiodic tasks and priority driven scheduling," *IEEE Transactions on Computers*, vol. 53, no. 3, 2004.

[2] J. Anderson, J. Calandrino, and U. C. Devi, "Real-time scheduling on multicore platforms," in *RTAS*, 2006.

[3] A. Bemporad and D. M. de la Pena, "Multiobjective model predictive control," *Automatica*, 2009.

[4] E. Bini, G. Buttazzo, and G. Buttazzo, "Rate monotonic analysis: The hyperbolic bound," *IEEE Transactions on Computers*, 2003.

[5] E. Bini, G. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *ECRTS*, 2005.

[6] E. Bini, G. C. Buttazzo, and M. Bertogna, "The multi supply function abstraction for multiprocessors," in *RTCSA*, 2009.

[7] A. Block *et al.*, "Adaptive multiprocessor real-time scheduling with feedback control," in *ECRTS*, 2008.

[8] B. Bui *et al.*, "Impact of cache partitioning on multi-tasking real time embedded systems," in *RTCSA*, 2008.

[9] A. Easwaran and B. Andersson, "Resource sharing in global fixed-priority preemptive multiprocessor scheduling," in *RTSS*, 2009.

[10] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-aware scheduling and analysis for multicores," in *EMSOFT*, 2009.

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

[12] D. Hardy *et al.*, "Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches," in *RTSS*, 2009.

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

[14] Kotera *et al.*, "Power-Aware Dynamic Cache Partitioning for CMPs," in *HIPEAC*, 2008.

[15] K. Lakshmanan *et al.*, "Partitioned fixed-priority preemptive scheduling for multi-core processors," in *ECRTS*, 2009.

[16] Y. Li *et al.*, "Timing analysis of concurrent programs running on shared cache multi-cores," in *RTSS*, 2009.

[17] J. Lin *et al.*, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *HPCA*, 2008.

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

[19] 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*, vol. 16, no. 6, 2005.

[20] J. M. Maciejowski, *Predictive Control with Constraints*. Prentice Hall.

[21] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *PACT*, 2008.

[22] V. Nelis and J. Goossens, "Mora: an energy-aware slack reclamation scheme for scheduling sporadic real-time tasks upon multiprocessor platforms," in *RTCSA*, 2009.

[23] F. Nemer, H. Casse, P. Sainrat, J.-P. Bahsoun, and M. D. Michiel, "Papabench: a free real-time benchmark," in *WCET*, 2006.

[24] M. Paolieri *et al.*, "Hardware support for wcet analysis of hard real-time multicore systems," in *ISCA*, 2009.

[25] J. Park *et al.*, "Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors," in *ISLPED*, 2010.

[26] J. Renau *et al.*, "SESC simulator," January 2005, http://sesc.sourceforge.net.

[27] 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*, 2008.

[28] V. Suhendra and T. Mitra, "Exploring locking & partitioning for predictable shared caches on multi-cores," in *DAC*, 2008.

[29] X. Wang, X. Fu, X. Liu, and Z. Gu, "Power-aware CPU utilization control for distributed real-time systems," in *RTAS*, 2009.

[30] J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," in *RTAS*, 2008.