TECS: Temperature- and Energy-Constrained Scheduling for Multicore Systems

Xiaoke Qin and Prabhat Mishra

Department of Computer and Information Science and Engineering
University of Florida, Gainesville FL 32611-6120, USA
{*xqin, prabhat*}*@cise.ufl.edu*

***Abstract*—The widespread use of multicore architectures with decreasing feature size is causing severe increase of on-chip power dissipation in modern embedded systems. This introduces both thermal and energy management problems that need to be addressed during the system level design. In this paper, we explore the DVS scheduling problem on multicore systems under both temperature and energy constraints. We present an exact algorithm as well as a polynomial time approximation scheme, since this problem is NP-hard. When the original problem is schedulable, our approximation algorithm is guaranteed to generate a solution, which will not violate the temperature constraint, and consume no more time or energy than a specified approximation bound, e.g., within 1%, of the optimal time consumption and energy constraints. We evaluate our approach using both real and synthetic benchmarks mapped on DVS capable multicore processors. The experimental results demonstrate that our technique is able to produce schedules close to the optimal solution with reasonable execution time.**

I. Introduction

Thermal management is becoming more and more important along with the performance improvement of modern multicore processors. Due to increasing integration of transistors into the same chip, the power densities are increasing dramatically. While the heat flux in desktop microprocessor is rising up to $250W/cm^2$, the peak power densities in embedded processors for handheld devices are also getting close to $100W/cm^2$. Dynamic voltage scaling (DVS) is a promising technique to address both the thermal and energy management problem for embedded systems. By exploiting the fact that the same task has different time consumption and power profile under different clock rate/voltage levels, DVS can reduce both peak temperature and total energy consumption effectively by running tasks at suitable voltage levels.

In this paper, we study the DVS scheduling problem on multicore processors under energy and temperature constraints. Since the task mapping and sequencing are already discussed in many existing works, in this paper, we focus on how to assign clock rate/voltage levels to tasks that are already mapped and sequenced on different cores, so that the total time consumption is minimized under both temperature and energy constraints. Our goal is to develop a Temperature and Energy Constrained Scheduling (TECS) for multicore systems. To avoid the state explosion problem, we propose an approximation scheme with polynomial time/space complexity based on the detailed analysis of the problem. Section II describes existing research efforts that focus on energy/temperature optimization in multicore systems. To the best of our knowledge, there are no prior works that consider both energy and temperature constraints in multicore systems and are guaranteed to produce schedules close to the optimal solution with reasonable execution time. The primary contribution of this paper is the development of an approximation scheme, which generates schedules in polynomial time when the tasks are schedulable. The resultant schedule is guaranteed to consume no more time or energy than a specified approximation bound. We have evaluated the effectiveness of our approach on both real and synthetic benchmarks.

The rest of the paper is organized as follows. Section II introduces relevant existing research works. Section III describes our system models. Section IV defines the TECS problem. Section V and Section VI discuss the optimal algorithm and our approximation scheme of the TECS problem in detail. Experimental results are presented in Section VII. Finally, Section VIII concludes the paper.

II. Related Work

Energy-aware scheduling with DVS has been widely studied to reduce the total energy consumption for real-time systems. For example, Aydin et al. [1] discussed the power-aware scheduling problems of periodic task sets. For multicore processors, Yang et al. [11] devised an approximation algorithm for energy optimized task mapping. Kolpe et al. [4] discussed efficient power management using clustered DVFS. *However, the peak temperature controlling problem is not addressed by these techniques.*

In recent years, the temperature-aware scheduling in real-time systems is becoming a quite attractive research topic. Zhang et al. [13] proved that the performance optimization problem under temperature constraint is NP-hard. They also introduced an approximation algorithm for the problem. In [12], the impact of leakage power are considered for the temperature constrained DVS problem. *Since these approaches focus only on temperature, the energy constraint is not considered.*

In [8], [9], the DVS scheduling problem under both energy and temperature constraints are studied in the framework of model checking, which may encounter the space explosion problem. Chantem et al. [2] proposed a very flexible framework to model the DVS scheduling problem in multicore processors using integer programming. They also presented several heuristics to reduce the constraint solving time. *Nevertheless, the optimality of the generated solution is not guaranteed.* In this paper, we propose a polynomial time

This work was partially supported by National Science Foundation (NSF) grants CNS-0746261 and CCF-1218629.

approximation scheme, which generates schedules that will not violate the temperature constraint and consume no more time or energy than a specified approximation bound, e.g., within 1%, of the optimal time consumption and energy constraints.

III. System Model

A. Processor Thermal Model

When the execution time of each task is long enough for the processor to reach the steady state temperature, we can use the matrix model [10] to calculate the steady state temperature on each core as

$$\boldsymbol{T}(t) = T_{amb} * \boldsymbol{I}(t) + \boldsymbol{C} * \boldsymbol{P}(t) \quad (1)$$

Here, T_{amb} is the ambient temperature, $\boldsymbol{C}$ is a $n \times n$ constant coefficient matrix, and $\boldsymbol{P}(t)$ is the power dissipation by each core under the clock rate assignment at time t.

B. Energy Model

We adopt the energy model proposed in [5]. Processor's dynamic power can be represented as $P_{dyn} = \alpha \cdot C \cdot V_{dd}^2 \cdot f$. Here V_{dd} is the supply voltage and f is the operation frequency. C is the total capacitance and α is the actual switching activity which varies for different applications. Static power is given by $P_{sta} = V_{dd} \cdot I_{subth} + |V_{bs}| \cdot I_j$ where V_{bs}, I_{subth} and I_j denote the body bias voltage, subthreshold current and reverse bias junction current, respectively. Hence, the overall power $P = P_{dyn} + P_{sta}$.

C. System Model

The system we consider can be modeled as:

- A multicore processor with M cores. Each core supports L discrete clock rate/voltage levels $\{f_1/v_1,\ f_2/v_2,\ \ldots,\ f_L/v_L\}$, where f_{min} is the lowest clock rate, and f_{max} is the highest.
- A set of n tasks, which has already been mapped and sequenced on different cores. We use τ_{ij} to denote the j^{th} task on core i. Let c_{ij} be the worst-case workload of τ_{ij}, and k_i be the total number of tasks mapped on core i. We also denote the total workload on core i by $w_i = \sum_{j=1}^{k_i} c_{ij}$.

For ease of discussion, the terms *task* and *job* refer to the same entity in the rest of this paper.

IV. Problem formulation

We assume that all tasks are already mapped and sequenced. A DVS schedule on a multicore system with task set $\{\tau_{ij} | 1 \le i \le M, 1 \le j \le k_i\}$ can be represented as a set of tuples $\{(r_{ij}, [t_{ij}, t'_{ij}]) | 1 \le i \le M, 1 \le j \le k_i\}\}$, where $(r_{ij}, [t_{ij}, t'_{ij}])$ means we execute τ_{ij} using clock rate r_{ij} during time interval $[t_{ij}, t'_{ij}]$. It is easy to see that clock rate switches always happen when some task finishes. When all tasks mapped to a core are finished, a core is turned off.

Given a set of n independent tasks $\{\tau_{ij} | 1 \le i \le M, 1 \le j \le k_i\}$, if the safe temperature threshold is C_T and the energy budget is C_E, TECS scheduling problem can be defined as follows.

Definition 1: **TECS** is formally defined as finding a multicore DVS schedule, R_{opt}, which minimize the total execution time, i.e., $\min \max_{1 \le i \le M} t'_{ik_i}$

subject to

$$\begin{aligned} & c_{ij}/r_{ij} \le t'_{ij} - t_{ij} \\ & \sum_{0 \le i \le n} P(r_{ij}) * (t'_{ij} - t_{ij}) \le C_E \\ & T(t) \le C_T, \forall t \ge 0 \\ & t'_{ij} \le t_{ij+1}, \forall j < k_i \end{aligned}$$

where $P(r_{ij})$ is power dissipation of a single core when task τ_{ij} is executing using clock rate r_{ij}. It can be proved that TECS problem is NP-hard. The proof is omitted due to page limit. It is available in the technical report [14].

Since the execution time of a typical task is long enough, the system will reach a steady state temperature. As a result, the peak steady temperature of cores is expected to occur only at clock rate switching point. Therefore, we do not need to calculate the transient temperature between clock rate switching points.

V. Optimal algorithm for TECS

The optimal solution of the TECS problem can be calculated using dynamic programming. The basic idea is to generate all possible execution paths of the system from the initial state. Notice that we consider inter-task DVS, i.e., the voltage switching is only allowed before the beginning of a new task. Any execution path of the system is uniquely determined by the system states at each switching point. Furthermore, since the number of cycles between two successive possible switching points can be estimated using remaining task workloads and clock rates on different cores, the state transition between different switching points can be performed as follows. Given a system state, we first identify the next task that is ready to execute. Next, we compute the system states at next switching point by executing this task with all possible clock rates. Finally, we mark the estimated state as a valid new state, if it does not violate the temperature or energy constraints.

Formally, given a task sequence on core i, at any time instant t, we define the progress of this task sequence as $p_i = w/w_i$, where $w_i = \sum_j c_{ij}$ is the total workload mapped on core i and w $(w \le w_i)$ is the completed workload on this core. The system status can be represented as a tuple $\boldsymbol{s} = (<p_1, r_1>, \ldots, <p_M, r_M>, E, t)$, where p_i and r_i are the current progress and clock rate of core i, respectively. E and t are the total energy and time consumption. The temperature of each core is not explicitly included in the system state tuple, because they can be calculated using the power of each core P_m and ambient temperature T_{amb} using Equation (1).

When some cores in the system are about to start execution of the next job in their task sequences, we encounter a potential clock rate switching point, or switching point for short. Since multiple cores can change clock rate at the same time, e.g., at $t = 0$, all possible clock rate assignments for M cores can be represented by a set of $M-$dimenional vectors. Formally, we define the set of possible clock **R**ate **A**ssignment $RA(\boldsymbol{s})$ for system state $\boldsymbol{s}$ as the direct product

$$RA(\boldsymbol{s}) = \bigotimes_{i=1}^{M} \begin{cases} \{0\} & \text{if } \boldsymbol{s}.p_i = 1 \\ \{f_1, \ldots, f_L\} & \text{else if } R_i(\boldsymbol{s}.p_i) = 0 \\ \{\boldsymbol{s}.r_i\} & \text{otherwise} \end{cases} \quad (2)$$

where

$$R_i(p_i) = \begin{cases} 0 & \text{if } p_i = 0 \\ \sum_{j=1}^{1} c_{ij}/w_i - p_i & \text{else if } \sum_{j=1}^{1} c_{ij}/w_i \geq p_i \\ \sum_{j=1}^{2} c_{ij}/w_i - p_i & \text{else if } \sum_{j=1}^{2} c_{ij}/w_i \geq p_i \\ \ldots & \\ \sum_{j=1}^{k_i} c_{ij}/w_i - p_i & \text{else if } \sum_{j=1}^{k_i} c_{ij}/w_i \geq p_i \end{cases} \quad (3)$$

$R_i(p_i)$ is the remaining progress until the beginning of next task on core i. $RA(\boldsymbol{s})$ returns a set of possible clock rate choices, which allows the core to choose from L voltage levels if it is about to start the next task, i.e., $R_i(p_i) = 0$. If all tasks on the same core are finished, i.e., $p_i = 1$, we shut down the core, by assigning clock rate 0. A core does not consume any more energy at clock rate 0.

In order to calculate the system state at next switching point, we define the state transition function $\boldsymbol{s'} = \boldsymbol{F}(\boldsymbol{s}, \boldsymbol{r})$ as

$$\begin{aligned} &\boldsymbol{s'}.p_i = \boldsymbol{s}.p_i + r_i * \delta / w_i \qquad \boldsymbol{s'}.r'_i = r_i,\ 1 \leq i \leq M \\ &\boldsymbol{s'}.E = \boldsymbol{s}.E + \sum_{i=1}^{M} P(r_i) * \delta \qquad \boldsymbol{s'}.t = \boldsymbol{s}.t + \delta \\ &\text{where } \delta = \min_{1 \leq i \leq M, p_i < 1} (R_i(\boldsymbol{s}.p_i + \sigma)) w_i / r_i \end{aligned} \quad (4)$$

σ is a very small positive number close to 0.

The state transition function $\boldsymbol{F}$ takes the system state at a switching point $\boldsymbol{s}$, and clock rate assignment vector $\boldsymbol{r}$ as inputs and produces the system state at the next switching point.

Algorithm 1 Exact solution to TECS

DPRA

1: $\mathcal{S} = \{\boldsymbol{s_1}\} = \{(<0,0>,...,<0,0>,0,0)\}$
2: **while** not all states in $\mathcal{S}$ are explored **do**
3: Pick an unexplored state $\boldsymbol{s}$ from $\mathcal{S}$ such that $\boldsymbol{s}$ contains at least one incomplete task sequence with the least progress among all states in $\mathcal{S}$
4: **for** each $\boldsymbol{r} \in RA(\boldsymbol{s})$ **do**
5: $\boldsymbol{s'} = \boldsymbol{F}(\boldsymbol{s}, \boldsymbol{r})$
6: **if** $\boldsymbol{r}$ violates temperature constraint C_T or $\boldsymbol{s'}.E > C_E$ **then**
7: continue
8: **if** $\exists \boldsymbol{s_0} \in \mathcal{S}$ s.t. $\boldsymbol{s_0}$ and $\boldsymbol{s'}$ agree on all values but time **then**
9: **if** $\boldsymbol{s_0}.t \leq \boldsymbol{s'}.t$ **then**
10: continue
11: **else**
12: $\mathcal{S} = \mathcal{S} - \{\boldsymbol{s_0}\}$ /*Remove $\boldsymbol{s_0}$*/
13: $\mathcal{S} = \mathcal{S} \bigcup \{\boldsymbol{s'}\}$ /*Add $\boldsymbol{s'}$*/
14: Find the state $\boldsymbol{s}_{opt}$ in $\mathcal{S}$ with the least time consumption, such that all tasks are finished. Construct the corresponding schedlue R_{opt} by backtracking from $\boldsymbol{s}_{opt}$ to $\boldsymbol{s_1}$.

Algorithm 1 shows the Dynamic Programming (DP) algorithm for clock Rate Assignment (DPRA) to obtain the optimal solution to the TECS problem. Initially, the set of system states $\mathcal{S}$ only contains a single state $\boldsymbol{s_1} = (<0,0>,...,<0,0>,0,0)$. During the DP process, we first pick $\boldsymbol{s} \in \mathcal{S}$, which contains at least one incomplete task sequence with the least progress among all states in $\mathcal{S}$. Suppose that there are m task sequences that are about to start new tasks. We try all possible combinations of clock rate assignments on these m cores, while keeping the clock rate unchanged on the rest $M-m$ cores. This will yield a set of assignments $RA(\boldsymbol{s})$, which contains L^m elements. Next, we calculate a system state $\boldsymbol{s'}$ based on $\boldsymbol{s}$ and clock rate assignment $\boldsymbol{r} \in RA(s)\}$. If $\boldsymbol{s'}$ does not violate any constraints, we add it to $\mathcal{S}$. The above process repeats until all states in $\mathcal{S}$ containing incomplete tasks are explored. Now, we need to find the state which has the least time consumption in $\mathcal{S}$.

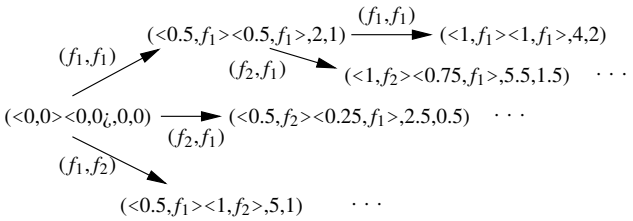


Fig. 1. State exploration in Algorithm 1

EXAMPLE 1: This example illustrates the flow of Algorithm 1 using a processor with $M = 2$ cores. Each of them have L=2 different clock rate levels, $f_1 = 100MHz$ and $f_2 = 200MHz$. Their power consumption are $P(f_1) = 1W$ and $P(f_2) = 4W$. There are three tasks $\tau_{1,1}$, $\tau_{1,2}$ and $\tau_{2,1}$ with workloads of 10^6, 10^6, and $2*10^6$ cycles, respectively. $\tau_{1,1}$ and $\tau_{1,2}$ are mapped to core 1, while $\tau_{2,1}$ is mapped to core 2. Therefore, we have $c_{1,1} = c_{1,2} = 10^6$, $c_{2,1} = 2*10^6$, $w_1 = c_{1,1} + c_{1,2} = 2*10^6$ and $w_2 = c_{2,1} = 2*10^6$. We choose the temperature constraint such that only one core can run at $200MHz$. We also choose $C_E = 10$J. When we apply Algorithm 1 to such a TECS instance, $\mathcal{S}$ contains only one element $\boldsymbol{s_1}$=(<0,0>,<0,0>,0,0) at the beginning. Thus, $\boldsymbol{s_1}$ is picked by line 3. Since we have $R_1(\boldsymbol{s_1}.p_1) = R_1(0) = 0$ and $R_2(\boldsymbol{s_1}.p_2) = 0$, the clock rates for both cores can be changed, i.e., $RA(\boldsymbol{s_1}) = \{f_1, f_2\} \otimes \{f_1, f_2\} = \{(f_1,f_1),(f_1,f_2),(f_2,f_1),(f_2,f_2)\}$ contains $L^M = 4$ elements, which represents four possible clock rate assignments. Next, we compute new states $\boldsymbol{s'}$ based on these assignments except (f_2,f_2), which violate the temperature constraint. If we pick $\boldsymbol{r} = (f_1,f_1)$, the new state $\boldsymbol{s_2} = \boldsymbol{F}(\boldsymbol{s_1}, \boldsymbol{r})$ can be computed as follows. First, we have $R_1(\boldsymbol{s_1}.p_1 + \sigma) = 0.5$, which means core 1 is $0.5w_1$ cycles far from the beginning of the next task $\tau_{1,2}$. Similarly, $R_2(\boldsymbol{s_1}.p_2 + \sigma) = 1$. Therefore, if we use clock rate $\boldsymbol{r} = (f_1,f_1)$, which makes both cores to run at $f_1 = 100MHz$, $\delta = \min(0.5w_1/f_1, w_2/f_1) = 1$sec. In other words, the next switching point will happen after 1sec. At that time, the progress values of core 1 and core 2 will be $\boldsymbol{s_2}.p_1 = 0 + f_1 * 1/w_1 = 0.5$ and $\boldsymbol{s_2}.p_2 = 0 + f_1 * 1/w_2 = 0.5$, respectively. We also compute the energy consumption $\boldsymbol{s_2}.E = 0 + P(f_1) * 1 + P(f_1) * 1 = 2$J and time consumption $\boldsymbol{s_2}.t = 1$sec. Therefore, the new state is $\boldsymbol{s_2}$ =(<0.5,f_1>,<0.5,f_1>,2,1). Since $\boldsymbol{s_2}$ and $\boldsymbol{r} = (f_1,f_1)$ do not violate any constraint, we add $\boldsymbol{s_2}$ into $\mathcal{S}$. We repeat this procedure until we find a state in $\mathcal{S}$, within which all tasks are finished with minimum total time consumption. Through backtracing, we can find the path that generates it: (<0,0>,<0,0>,0,0)→ (<0.5,f_1>,<1,f_2>,5,1)→ (<1,f_2>,<1,f_1>,7,1.5). The corresponding scheduling R_{opt} is (<f_1,0,1>,<f_2,1,1.5>,<f_2,0,1>), which means $\tau_{1,1}$, $\tau_{1,2}$ and $\tau_{2,1}$ should be executed using f_1, f_2, and f_2, respectively. □

The time and space complexity of the exact algorithm is $O(L^n)$, becuase each of the n tasks can be executed at L different voltage levels.

VI. APPROXIMATION ALGORITHM FOR TECS

The basic idea of our approximation algorithm is built on discretization of the state space. The space size is reduced

by rounding up all values in the state vector, and by merging states that agree on all values after rounding. Unfortunately, in TECS problem, this method cannot be applied directly to progress values. Recall that we define the progress of a task sequence on each core to represent how many instruction or workload has already been completed. Rounding up progress values introduces two problems. First, the switching points, which are defined based on progress values may be skipped, because they usually do not coincide with the discretized progress values. Second, the rounding operation essentially means we skip some instructions without executing them. Therefore, if we apply the obtained scheduling in reality, the actual progress will not match with the ones we calculated in dynamic programming. As a result, the temperature or energy constraints may be violated.

In this paper, we solve both problems as follows. First, we view a state $\boldsymbol{s} \in \mathcal{S}$ not as a real system state, but a pessimistic approximation of a real system state. Second, we insert a suitable "idle time" at each switching point, so that the difference between the real execution and estimated value in dynamic programming can be bounded. In this way, we can obtain an approximated estimation of the actual execution under any clock rate selections. Before we introduce our approximation scheme, we first introduce the modified version of the state transition function and clock rate assignment function, which are used to build the approximation algorithm. The modified state transition function $\boldsymbol{s'} = \boldsymbol{F}_{\Delta_t}(\boldsymbol{s}, \boldsymbol{r})$ is defined as

$$\begin{aligned}
&\boldsymbol{s'}.p'_i = \boldsymbol{s}.p_i \text{ if } r_i = f_I \ ; \quad \boldsymbol{s}.p_i + r_i * \delta / w_i, \text{otherwise} \\
&\boldsymbol{s'}.r'_i = \boldsymbol{s}.r_i \text{ if } r_i = f_I \ ; \quad r_i, \text{otherwise} \\
&\boldsymbol{s'}.E = \boldsymbol{s}.E + \sum_{i=1}^{M} P(r_i) * (\delta + 2\Delta_t) \\
&\boldsymbol{s'}.t = \boldsymbol{s}.t + \delta + 2\Delta_t \\
&\text{where } \delta = \min_{1 \le i \le M, p_i < 1} R_i(\boldsymbol{s}.p_i + \sigma) * w_i / r_i \\
&\qquad \sigma \text{ is a very small positive number close to 0.}
\end{aligned} \tag{5}$$

An extra increment ($2\Delta_t$) is added, which represents the "idle time". $RA_\varepsilon(\boldsymbol{s})$ is the modified version of $RA(\boldsymbol{s})$, which is defined as

$$RA_\varepsilon(\boldsymbol{s}) = \bigotimes_{i=1}^{M} \begin{cases} \{0\} & \text{if } \boldsymbol{s}.p_i = 1 \\ \{f_1, ..., f_L\} & \text{else if } R_i(\boldsymbol{s}.p_i) \le \Delta_P \\ \{\boldsymbol{s}.r_i\} & \text{otherwise} \end{cases} \tag{6}$$

Algorithm 2 shows the details of our approximation algorithm $DPRA_\varepsilon$, where $\boldsymbol{h}$ in line 9, is a partial rounding up function $\boldsymbol{s'} = \boldsymbol{h}(\boldsymbol{s})$. It is defined as

$$\begin{aligned}
&\boldsymbol{s'}.p_i = \lceil \boldsymbol{s}.p_i / \Delta_p \rceil * \Delta_p \quad \boldsymbol{s'}.r_i = \boldsymbol{s}.r_i, \ i = 1, ..., M \\
&\boldsymbol{s'}.E' = \lceil \boldsymbol{s}.E / \Delta_E \rceil * \Delta_E \quad \boldsymbol{s'}.t' = \boldsymbol{s}.t
\end{aligned} \tag{7}$$

We first compute the "step size" Δ_E, Δ_P and Δ_t for each constraint based on the value of ε. After that, $DPRA_\varepsilon$ parallels the exact algorithm *DPRA* except that the progress and energy values in each new system state $\boldsymbol{s}$ is rounded up to the next available discretized value. This is achieved by applying function $\boldsymbol{h}$, which forces the progress and energy value of the resultant state to be an integer multiple of Δ_P or Δ_E. For example, suppose we have $\Delta_P = 0.1$ and $\Delta_E = 0.2$, a new state $\boldsymbol{F}_{\Delta_t}(\boldsymbol{s}, \boldsymbol{r}))$ =(<0.5,f_2>, <0.25,f_1>,1,2.5,0.5) will be recorded as

Algorithm 2 Approximation algorithm of TECS

$DPRA_\varepsilon$

1: $\Delta_E = \varepsilon * C_E / 4n$
2: $\mu = \max_{1 \le i \le M} w_i / f_{min}$
3: $\Delta_P = \min(\Delta_E / \mu P_{max}, \varepsilon * f_{min} / (f_{max} * 2n))$. P_{max} is the maximum power dissipation of the entire processor.
4: $\Delta_t = \Delta_P * \mu$
5: $\mathcal{S} = \{\boldsymbol{s_1}\} = \{(<0,0>, ..., <0,0>, 0, 0)\}$
6: **while** not all states in $\mathcal{S}$ are explored **do**
7: Pick an unexplored state $\boldsymbol{s}$ from $\mathcal{S}$ such that $\boldsymbol{s}$ contains at least one incomplete task sequence with the least progress among all states in $\mathcal{S}$
8: **for** each $\boldsymbol{r} \in RA(\boldsymbol{s})$ **do**
9: $\boldsymbol{s'} = \boldsymbol{h}(\boldsymbol{F}_{\Delta_t}(\boldsymbol{s}, \boldsymbol{r}))$
10: **if** $\boldsymbol{r}$ violates temperature constraint C_T or $\boldsymbol{s'}.E > (1+\varepsilon)C_E$ **then**
11: continue
12: **if** $\exists \boldsymbol{s_0} \in \mathcal{S}$ s.t. $\boldsymbol{s'}$ and $\boldsymbol{s_0}$ agree on all values but time **then**
13: **if** $\boldsymbol{s_0}.t \le \boldsymbol{s'}.t$ **then**
14: continue
15: **else**
16: $\mathcal{S} = \mathcal{S} - \{\boldsymbol{s_0}\}$
17: $\mathcal{S} = \mathcal{S} \bigcup \{\boldsymbol{s}\}$
18: Find the state $\boldsymbol{s}_{apx}$ in $\mathcal{S}$ with the least time consumption OPT_{apx}, such that all tasks are finished. Construct the corresponding schedule R_{apx} by backtracking from $\boldsymbol{s}_{apx}$ to $\boldsymbol{s_1}$. If a task is skipped due to rounding, it is scheduled as a part of the previous task on the same core.

$\boldsymbol{h}(\boldsymbol{F}_{\Delta_t}(\boldsymbol{s}_0, \boldsymbol{r})))$
=(<$\lceil 0.5/0.1 \rceil$*0.1,f_2>,<$\lceil 0.25/0.1 \rceil$*0.1,f_1>,$\lceil 2.5/0.2 \rceil$*0.2,0.5)
=(<0.5,f_2>,<0.3,f_1>,2.6,0.5)

The correctness of our proposed algorithm is guaranteed by the following theorem.

Theorem 6.1: Given a *TECS* instance I, if I is schedulable with optimal time consumption OPT, $DPRA_\varepsilon(I)$ will return a schedule in polynomial time, which does not violate the temperature constraint, while its energy and time consumption are at most $(1+\varepsilon)C_E$ and $(1+\varepsilon)OPT$, respectively.

To prove the theorem, we need to prove following three lemmas. First, if $DPRA_\varepsilon$ finds a schedule, it satisfies all the constraints in any real executions with time consumption at most OPT_{apx} (Lemma 6.1). Next, if the optimal schedule exists, $DPRA_\varepsilon$ always returns a schedule, which is close to the optimal one (Lemma 6.2). Finally, $DPRA_\varepsilon$ is a polynomial time algorithm (Lemma 6.3).

Lemma 6.1: Given a *TECS* instance I, if $DPRA_\varepsilon(I)$ finds a schedule R_{apx} with estimated time consumption OPT_{apx}, we show that R_{apx} is a feasible schedule of I, whose actual time consumption does not exceed OPT_{apx}.

Proof: Since R_{apx} is found by $DPRA_\varepsilon(I)$, $\mathcal{S}$ must be a state $\boldsymbol{s}_{apx}$ and a path with K states $\boldsymbol{s_1} \to ... \to \boldsymbol{s}_{K-1} \to \boldsymbol{s_K}(\boldsymbol{s}_{apx})$.

When R_{apx} is applied in reality, we apply the clock rate assignment $\boldsymbol{s_l}.r_i$ at time $\boldsymbol{s_l}.t$ for $1 \le l \le K$. When the current job on a core is finished, we keep the core running idle job until the switching point, where the next task is scheduled to

start. To prove this lemma, we need to show that 1) all jobs have enough time to finish, and 2) all constraints are met.

The first statement can be proved by showing that each job has enough time to run. Suppose a task τ on core i starts from the l^{th} switching point, i.e., state. If τ finishes at the m^{th} switching point, i.e., $\boldsymbol{s_l}$, the next task on the same core starts from the m^{th} switching point, i.e., $\boldsymbol{s_m}$, the time allocated for this task is $\boldsymbol{s}_m.t - \boldsymbol{s}_l.t$. Since we perform $m-l$ rounds of computation to obtain $\boldsymbol{s}_m$ from $\boldsymbol{s}_l$, there can be at most $m-l$ rounding up during the calculation from $\boldsymbol{s_l}.p_i$ to $\boldsymbol{s_m}.p_i$. Therefore,

$$\boldsymbol{s}_m.t \geq \boldsymbol{s}_l.t + (\boldsymbol{s}_m.p_i - \boldsymbol{s}_l.p_i - (m-l)\Delta_p) * w_i/\boldsymbol{s}_l.r_i + 2(m-l)\Delta t$$

Notice that $\Delta_t = \Delta_P * \mu \geq \Delta_P * w_i/r_i$, $m > l$

$$\begin{aligned}\boldsymbol{s}_m.t - \boldsymbol{s}_l.t &\geq (\boldsymbol{s}_m.p_i - \boldsymbol{s}_l.p_i + (m-l)\Delta_p) * w_i/r_i \\ &\geq (\boldsymbol{s}_m.p_i - \boldsymbol{s}_l.p_i + \Delta_p) * w_i/r_i\end{aligned}$$

However, the workload of τ can be at most $(\boldsymbol{s}_m.p_i - \boldsymbol{s}_l.p_i + \Delta_p) * w_i$. For example, suppose the exact progress of task τ is 0.501 to 0.699 and $\Delta_P = 0.1$. After rounding, we have $\boldsymbol{s}_m.p_i = 0.6$ and $\boldsymbol{s}_l.p_i = 0.7$. Clearly, the total workload of τ is not more than $(\boldsymbol{s}_m.p_i - \boldsymbol{s}_l.p_i + \Delta_p) * w_i = 0.2w_i$. Therefore, all tasks will have enough time for execution when R_{apx} is applied.

Now we prove that all constraints are met by considering the following relations among different successive states on path $\boldsymbol{s}_1 \rightarrow \boldsymbol{s}_2 \rightarrow ... \rightarrow \boldsymbol{s}_{K-1} \rightarrow \boldsymbol{s}_K$.

$$\begin{aligned}\boldsymbol{s}_2 &= \boldsymbol{h}(\boldsymbol{F}_{\Delta_t}(\boldsymbol{s}_1, \boldsymbol{r}_1)) \\ &... \\ \boldsymbol{s}_K &= \boldsymbol{h}(\boldsymbol{F}_{\Delta_t}(\boldsymbol{s}_{K-1}, \boldsymbol{r}_K - 1))\end{aligned} \tag{8}$$

Based on the logic of $DPRA_\varepsilon(I)$, it is easy to see that $(1+\varepsilon)C_E \geq \boldsymbol{s}_l.E$ holds for $1 \leq l \leq K$. Let the state transition path produced by R_{apx} during a real execution be $\boldsymbol{s}_1 \rightarrow \boldsymbol{s}'_2 \rightarrow ... \rightarrow \boldsymbol{s}'_{K-1} \rightarrow \boldsymbol{s}'_K$. Clearly, we have

$$\begin{aligned}\boldsymbol{s}'_2 &= \boldsymbol{F}_{\Delta_t}(\boldsymbol{s}'_1, \boldsymbol{r}_1) \\ &... \\ \boldsymbol{s}'_K &= \boldsymbol{F}_{\Delta_t}(\boldsymbol{s}'_{K-1}, \boldsymbol{r}_{K-1})\end{aligned} \tag{9}$$

Since all components of vector functions $\boldsymbol{h}$ are increasing functions, i.e., $\boldsymbol{x} \geq \boldsymbol{y} \Rightarrow \boldsymbol{h}(\boldsymbol{x}) \geq \boldsymbol{h}(\boldsymbol{y})$[1], we can verify that $\boldsymbol{s}_2 \geq \boldsymbol{s}'_2$,...,$\boldsymbol{s}_{K-1} \geq \boldsymbol{s}'_{K-1}$ and $\boldsymbol{s}_{apx} \geq \boldsymbol{s}'_K$. Therefore,

$$\begin{aligned}(1+\varepsilon)C_E &\geq \boldsymbol{s}'_k.E \\ OPT_{apx} = \boldsymbol{s}_{apx}.t = \boldsymbol{s}_K.t &\geq \boldsymbol{s}'_K.t\end{aligned} \tag{10}$$

Notice that temperature and energy values change monotonically between $\boldsymbol{s}'_k$ and $\boldsymbol{s}'_{k+1}$ during real execution. Equation (10) ensures that all constraints are met. ■

Lemma 6.2: Given a *TECS* instance I, if I is schedulable with optimal time consumption OPT, $DPRA_\varepsilon(I)$ returns a schedule R_{apx} with estimated time consumption $OPT_{apx} \leq (1+\varepsilon)OPT$.

This lemma ensures if the task set is schedulable, $DPRA_\varepsilon(I)$ always finds a schedule. Due to the page limit, the proof of this lemma is available in the technical report [14].

Lemma 6.3: $DPRA_\varepsilon$ is a polynomial time algorithm with the number of tasks, n.

[1] $\boldsymbol{x} \geq \boldsymbol{y}$ means each component of vector $\boldsymbol{x}$ is larger or equal to its corresponding component in vector $\boldsymbol{y}$

Proof: To verify $DPRA_\varepsilon$ is a polynomial time algorithm, we first show that the number of states in $\mathcal{S}$ is $O((n/\varepsilon)^{M+1})$. It is easy to see that the energy value is discretized into $4n/\varepsilon$ different values. For progress values, there are $1/\Delta_P$ different values allowed for each core. If $\Delta_E/\mu P_{max} < \varepsilon * f_{min}/(2nf_{max})$,

$$\frac{1}{\Delta_P} = \mu\frac{P_{max}}{\Delta_E} = \max_{1\leq i\leq M} w_i \frac{4nP_{max}}{\varepsilon C_E f_{min}}$$

Let the most energy efficient clock rate and the corresponding power of a single core be f_e and $P(f_e)$. Clearly, we can safely assume $C_E \geq P(f_e) * \max_{1\leq i\leq M} w_i/f_e$. Otherwise the workload $\max_{1\leq i\leq M} w_i$ cannot be finished within C_E and there is no need to run $DPRA_\varepsilon$. Therefore,

$$\frac{1}{\Delta_P} \leq \frac{4nf_eP_{max}}{\varepsilon f_{min}P(f_e)} \leq \frac{4nMf_eP(f_{max})}{\varepsilon f_{min}P(f_e)}$$

If $\Delta_E/\mu P_{max} \geq \varepsilon * f_{min}/(2nf_{max})$,

$$\frac{1}{\Delta_P} = \frac{2nf_{max}}{f_{min}\varepsilon}$$

In either case, $1/\Delta_P$ is no more than n/ε times a constant. Both $P(f_{max})/P(f_e)$ and f_{max}/f_{min} are normally less than 10. Therefore, there are at most $O((n/\varepsilon)^{M+1})$ states in $\mathcal{S}$. At the same time, the number of different voltage assignments we can choose, i.e., the size of $RA_\varepsilon(\boldsymbol{s})$, is also no more than $(L+1)^M$, which is a constant. Therefore, the overall complexity of $DPRA_\varepsilon$ is $O((n/\varepsilon)^{M+1})$. ■

VII. Experiments

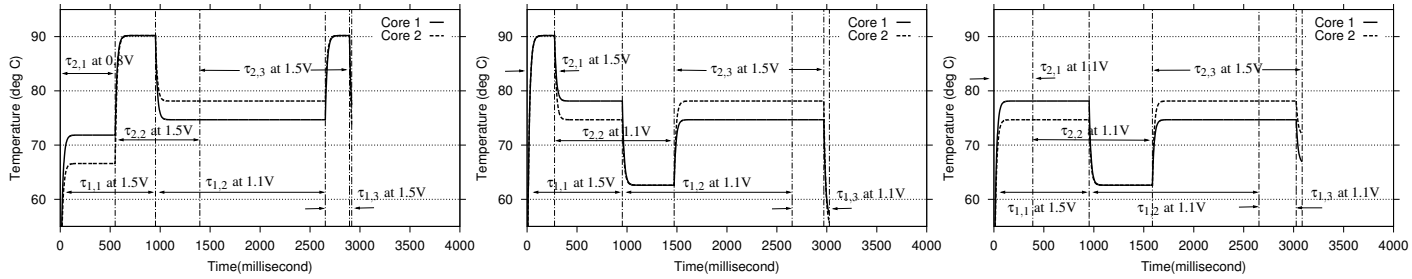
A. Experimental Setup

The experiments were conducted on 2 core, 4 core, and 6 core processors. Each core is abstracted as a $8mm \times 8mm$ square. The cores are arranged in 2×1, 2×2 and 3×2 meshes, respectively. We model each core as a DVS-capable processing unit with three voltage/frequency levels (1.5V-206MHz, 1.1V-133MHz, and 0.8V-103MHz) like StrongARM [6]. We choose some tasks from the Mibench and obtain the workload (worst case cycle numbers) from M5 simulator. We also use synthetic task sets which are randomly generated with each of them having execution time in the range of 500 - 5000 milliseconds. We adopt the approach in [10] to compute the steady state temperature. The ambient temperature and initial temperature of the processor are set to $32^\circ C$ and $40^\circ C$, respectively. The exact and approximation algorithms are implemented in C++. All experiments were performed on 3GHz workstation with 20GB RAM.

B. Results on real benchmarks

We choose 6 jobs from MiBench [3], including algorithms from communication (*FFT, CRC32*), security (*sha*), sound compression (*untoast*), and automotive (*basicmath, qsort*). The workload of these jobs were in range of $5*10^7 - 3*10^8$ cycles. We use the exact algorithm DPRA to schedule these tasks on 2 core processor. *CRC32* ($\tau_{1,1}$), *qsort* ($\tau_{1,2}$), and *untoast* ($\tau_{1,3}$) are mapped to core 1. *sha* ($\tau_{2,1}$), *FFT* ($\tau_{2,2}$), and *basicmath* ($\tau_{2,3}$) are mapped to core 2. We depict the temperature curves of each core in Figure 2, when different temperature and energy constraints are applied.

In Figure 2a, the temperature constraint is not violated when both cores run at 1.5V. DPRA schedules jobs on different cores to execute using the maximum voltage level at the same



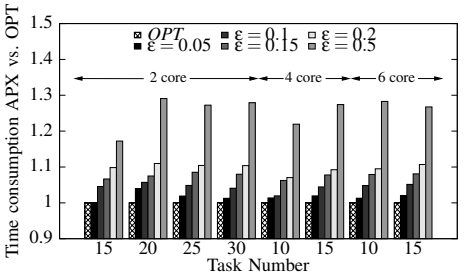
(a) $C_T = 95°C$ and $C_E = 23000mJ$ (b) $C_T = 95°C$ and $C_E = 22000mJ$ (c) $C_T = 85°C$ and $C_E = 23000mJ$

Fig. 2. Temperature and energy constrained scheduling

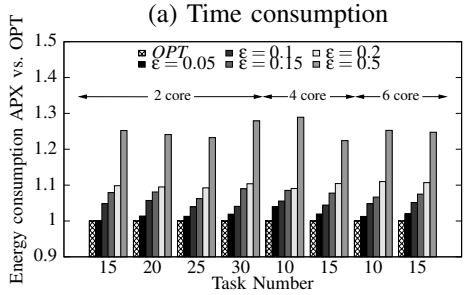
time, i.e., task $\tau_{1,1}$ and $\tau_{2,2}$, to minimize the time consumption. When the energy budget reduces, tasks with large workload is executed using lower voltage level to save energy as shown in Figure 2b. As we can see, $\tau_{2,2}$ is executed using 1.1V instead of 1.5V, when the energy budget reduces to 22000mJ. Similarly, when the temperature constraint becomes tighter, less number of tasks are executed using the maximum voltage level to decrease the peak temperature. As shown in Figure 2c, two cores no longer run using 1.5V at the same time. Although the energy budget is still sufficient, the time consumption increases slightly compared to Figure 2a.

C. Results on synthetic benchmarks

We evaluated the performance of our approximation scheme using task sets with different sizes. Figure 3a and 3b show the actual ratio between the approximation results (APX) and the optimal solution (OPT) for time and energy consumption, respectively. It can be seen that the actual ratio is usually smaller than the expected ratio $1+\varepsilon$. For example, for $\varepsilon = 0.02$, it is expected to produce results within 2% of the optimal values. The actual gap between the optimal solution and the approximation scheduling is significantly less than 2%.



(a) Time consumption



(b) Energy consumption

Fig. 3. Accuracy of $DPRA_\varepsilon$.

We also evaluated the running time of our approximation scheme with different ε and number of tasks. The results on 2 core and 4 core systems are shown in Figure 4. Curve *DPRA* represents the execution time of the exact algorithm *DPRA*, which grows exponentially with the number of tasks. As expected, $DPRA_\varepsilon$ requires more time for smaller ε or larger job set size but always significantly smaller than the exact algorithm *DPRA*.

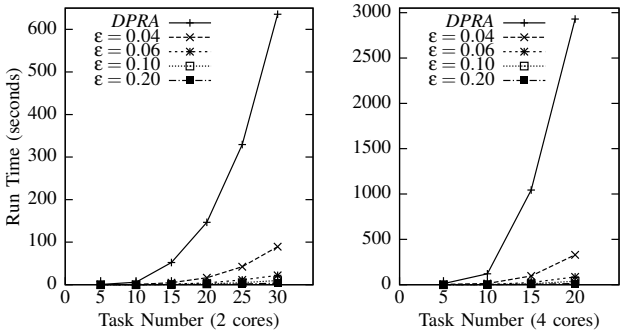


Fig. 4. Running time with different job set size and ε.

VIII. Conclusion

In this paper, we studied task scheduling problem on a multicore processor with DVS capability under both temperature and energy constraints. We presented a polynomial time approximation scheme. When the original problem is schedulable, our approximation algorithm is guaranteed to generate a solution, which will not violate the temperature constraint, and consume no more time or energy than a specified approximation bound, e.g., within 1%, of the optimal time consumption and energy constraints. We evaluated our approach using both real and synthetic benchmarks mapped on real multicore processors. The experimental results demonstrated that our technique is able to produce schedules close to optimal solution with reasonable execution time.

References

[1] H. Aydin et al. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Comput.*, 53(5):584–600, 2004.

[2] T. Chantem et al. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. In *Proc. DATE*, 2008.

[3] M. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *Proc. IEEE WWC*, 2001.

[4] T. Kolpe, et al, Enabling Improved Power Management in Multicore Processors through Clustered DVFS. In *Proc. DATE*, 2011.

[5] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proc. ICCAD*, 2002.

[6] *Marvell StrongARM 1100 processor*. www.marvell.com.

[7] U. of Virginia. http://lava.cs.virginia.edu/hotspot/.

[8] X. Qin et al. TCEC: Temperature and Energy-Constrained Scheduling in Real-Time Multitasking Systems. *IEEE Trans. on CAD.*, 31(8): 1159–1168, 2012.

[9] W. Wang et al. Temperature-and energy-constrained scheduling in multitasking systems:A model checking approach. In *ISLPED*, 2010.

[10] Z. Wang and S. Ranka. A simple thermal model for multi-core processors and its application to slack allocation. In *Proc. IPDPS*, 2010.

[11] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In *DATE*, 2009.

[12] L. Yuan and G. Qu. Alt-dvs: Dynamic voltage scaling with awareness of leakage and temperature for real-time systems. In *AHS*, 2007.

[13] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature aware scheduling problem. In *ICCAD*, 2007.

[14] X. Qin and P. Mishra. Multicore scheduling with temperature and energy constrains. CISE Technical Report, 2011.