Temperature-aware Task Partitioning for Real-Time Scheduling in Embedded Systems*

Zhe Wang, Sanjay Ranka and Prabhat Mishra
Dept. of Computer and Information Science and Engineering
University of Florida, Gainesville, USA
Email: {zhwang, sanjay, prabhat}@cise.ufl.edu

***Abstract*—Both power and heat density of on-chip systems are increasing exponentially with Moore's Law. High temperature negatively affects reliability as well the costs of cooling and packaging. In this paper, we propose task partitioning as an effective way to reduce the peak temperature in embedded systems running either a set of periodic heterogeneous tasks with common period or periodic heterogeneous tasks with individual period. For task sets with common period, experimental results show that our task partitioning algorithms is able to reduce the peak temperature by as much as 5.8^oC as compared to algorithms that only use task sequencing. For task sets with individual period, EDF scheduling with task partitioning can also lower the peak temperature, as compared to simple EDF scheduling, by as much as 6^oC. Our analysis indicates that the numbers of additional context switches (overhead) is less than 2 per task, which is tolerable in many practical scenarios.**

I. Introduction

The power density of on-chip systems has doubled every three years and this rate is expected to increase as frequencies scale faster than operating voltages [1]. This makes thermal management a significant design challenge for microprocessors. Rise in on-chip temperature directly impacts the performance and time-to-failure of switching devices. This is accentuated by the fact that the cost of cooling increases super-linearly with the rise of power consumption [2]. Also, the cost of cooling and packaging is one of the significant contributors to computer system [3]. High temperature increases leakage power of the chip, and thereby potentially lead to thermal runaway. It has been shown that a reduction in peak temperature of 10-15^oC can double the lifetime of the chip [4].

Existing power-aware techniques do not address the temperature issues in embedded systems. The main reason is that the power distribution of multiprocessors is not uniform. Localized heating occurs much faster than chip-wide heating, leading to nonuniform temperature distribution on the chip with localized high-temperature hot spots and spatial gradients [5]. Traditional methods to control the on-chip temperature is to employ better packaging and cooling techniques (e.g., active fan cooling, water cooling and heat pipe). These active cooling systems may not always be suitable for embedded systems because of the space and battery limitations. Building thermal analysis ability into EDA flow allows the system to address the impact of temperature on various on-chip parameters and incorporate effects of non-uniform thermal profiles during IC design process. However, such technologies are unable to deal with a variety of runtime situations.

In this paper, we focus on software approaches for thermal management. These approaches are flexible and do not have some of the limitations that are described above. The processor thermal behavior can be effectively modeled using RC model [5]. If the average power of a processor is P over a time period t, then the transient temperature $T(t)$ at the end of this period, using this model, is given by:

$$T(t) = P \times R + T_A - (P \times R + T_A - T_i)e^{-t/RC} \quad (1)$$

where R is the thermal resistance and C is the thermal capacitance, T_A is the ambient temperature and T_i is the initial temperature.

Given a particular chip and its outside environment, ambient temperature, thermal resistance and capacitance are fixed. Based on the parameters of Equation (1), there are three major factors affecting the on-chip transient temperature: average power of the processor, initial temperature and execution time. Dynamic Voltage and Frequency Scaling (DVFS) can be used to reduce the power consumption by lowering the supply voltage and operating frequency, thereby reduce the on-chip temperature [6]–[11]. However, DVFS faces a serious problem in time-constrained applications. Temperature aware task sequencing algorithm, which reduces the initial temperature, developed in [12] can reduce peak temperature compared to a random sequence. However, temperature aware task sequencing fails to reduce temperature in cases when one or more of the "hot" [1] tasks are long. The algorithm to defer execution of hot tasks [13] fails to reduce temperature in the same situation. This is because when the execution time of a "hot" task is too long, it can lead to a high steady-state temperature irrespective of the initial temperature.

We propose to partition the "hot" tasks into multiple subtasks and interleave these subtasks with "cool" tasks to reduce the overall maximum temperature (see Figure 2). The focus of this paper is to use this technique effectively to reduce the maximum temperature. To the best of our knowledge, our work is the first attempt to develop efficient task partitioning algorithms to demonstrate significant temperature reduction. In this paper, we propose a heuristic task partitioning algorithm using "cool" tasks to interleave "hot" tasks for a periodic set of tasks with common period. We also propose another heuristic task partitioning algorithm for a periodic set of tasks with individual period. Finally, we provide a thorough evaluation and comparison to show how task partitioning can assist in thermal-aware management problems. Experimental results show that our algorithm outperforms the task sequencing algorithm [12] by reducing the peak temperature by as much as 6^oC.

The rest of the paper is organized as follows. Section II describes the background of thermal aware analysis. Section III introduces the problems of periodic tasks with common and individual period, and proposes heuristic task partitioning algorithms to solve them. Section IV compares these algorithms with task sequencing algorithm and EDF algorithm, respectively. Section V concludes the paper.

II. Preliminaries

In this section, we briefly describe three related concepts. First, we discuss how to measure the thermal profile of a task. Next,

*This work was partially supported by NSF grant CCF-0903430 and SRC grant 2009-HJ-1979.

[1] We define "hot" tasks as tasks with higher average power consumption, and "cool" tasks as tasks with lower average power consumption.

we present how to analyze the thermal profile of a task sequence. Finally, we define the peak temperature of a task sequence.

A. Thermal profile of individual tasks

The basic thermal equation has already been introduced in Equation (1). By letting $t \to \infty$ in Equation (1), we can get the steady-state temperature:

$$T_S = P \times R + T_A \tag{2}$$

Based on our experiments, it takes less than 1 second to reach steady-state temperature for a $1.5GHz$ processor with product of thermal resistance and thermal capacitance to be $0.2053\ Joules/Watt$.

B. Thermal profile of task sequences

Consider a periodic set of heterogeneous tasks (i.e., tasks with different thermal profiles), with the execution time of these tasks given by $c_1, c_2, ..., c_N$. The periods of these tasks are given by $p_1, p_2, ..., p_N$. The average power consumption during execution time is given by $P_1, P_2, ..., P_N$. Suppose these tasks are ordered in a particular sequence $S =< \tau_1, ..., \tau_N >$. The hyper-period of these tasks is defined by $LCM_{i=1}^{N}\{p_i\}$, where LCM stands for the least common multiple. By executing these tasks in a hyper-period for a larger number of iterations (they are periodic tasks), the temperature of these tasks will rise from initial temperature and reach a final temperature, where the thermal profile of the hyper-period exhibits a recurring pattern [12]. We call this steady-state the hyper-period steady-state. Using the above arguments, we can analyze the thermal profile of one hyper-period sequence, other hyper-period sequences will have exactly the same thermal profile as this one.

C. Peak temperature of task sequences

Using the above simplifications, we have the final temperature of each task as follows:

$$\begin{aligned} T_1 &= P_i \times R + T_A - (P_i \times R + T_A - T_N)e^{-c_1/RC} \\ &\cdots \\ T_N &= P_i \times R + T_A - (P_i \times R + T_A - T_{N-1})e^{-c_1/RC} \end{aligned} \tag{3}$$

As we can see, the temperature in Equation (1) is a monotonic function, when $P_i \times R + T_A > T_i$, the temperature of the processor increases during the task execution time and vice versa. Therefore, either T_i or $T(t)$ is the maximum temperature during the execution time of the task. Thus, we define the maximum final temperature of tasks in one hyper-period as the peak temperature of the sequence.

$$\text{peak temperature} = \max\{T_1, T_2, \cdots, T_N\} \tag{4}$$

III. Task partitioning algorithms

There are two major challenges developing algorithms that use task partitioning to reduce peak temperature:

1) *Number of Partitions:* A task can be partitioned into a large number of very small pieces. However, this may result in significant overhead of preemption and restart. Choosing the right number of partitions that carefully tradeoffs the number of partitions and the resultant temperature reduction is important.
2) *Sequencing of Subtasks:* A reordering of "hot" and "cool" tasks has to ensure that the subtasks of a given task maintain the sequential order. For example if a task A is decomposed into subtasks A1, A2 and A3. A1 should always be executed before A2, and A2 should always be executed before A3.

Besides the obvious novelty of proposing a partitioning approach for addressing the thermal issues, the paper develops novel algorithms to address the following two broad scenarios:

1) A periodic set of tasks with common period. All the tasks have the same arrival time and deadline.
2) A set of periodic tasks with individual period. Each task may have different arrival time and deadline.

In this section, we first give an illustrative example showing that the peak temperature of task partitioning algorithm is less than that of task sequencing algorithm [12]. Assume that two tasks, τ_1 and τ_2, have average power consumption P_1 and P_2, respectively. Their execution times are t_1 and t_2, $t_1, t_2 > 0$. Without loss of generality, we assume $P_1 < P_2$. Therefore, τ_1 is a "cool" task and τ_2 is a "hot" task. Based on the task sequencing algorithm [12], as Figure 1 shows, the "hot" task is followed by a "cool" task. The temperature at time t is denoted as $T(t)$, $t \in [0, t_1 + t_2]$. Thus, the initial temperature is $T(0)$. The ambient temperature is T_A.

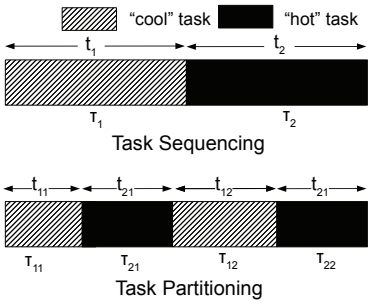

Fig. 1. An example of the task sequencing and task partitioning. There are two tasks in task sequencing: a "cool" task τ_1 followed by a "hot" task τ_2. The execution time of τ_1 and τ_2 are t_1 and t_2, respectively. These two tasks are partitioned into four subtasks in task partitioning. The (task, execution time) sequence is (τ_{11}, t_{11}), (τ_{21}, t_{21}), (τ_{12}, t_{12}), (τ_{22}, t_{22}), where $t_1 = t_{11} + t_{12}$, $t_2 = t_{21} + t_{22}$.

We also introduce the task partitioning. We assume that both task τ_1 and τ_2 are partitioned into two subtasks. By interleaving the subtasks, the (task, execution time) sequence is (τ_{11}, t_{11}), (τ_{21}, t_{21}), (τ_{12}, t_{12}), (τ_{22}, t_{22}) (see Figure 1). Figure 2 shows the transient temperature of task sequencing and task partitioning. We can see that task partitioning can achieve lower peak temperature because the "cool" task absorbs the heat generated by the "hot" task.

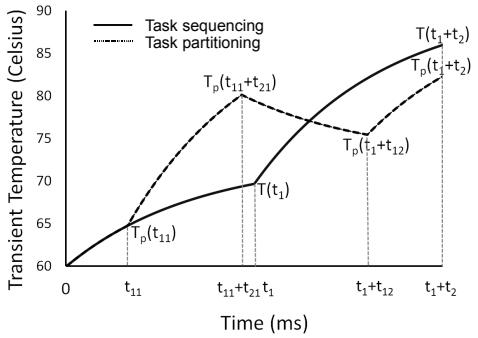

Fig. 2. Transient temperature comparison between the task sequencing and task partitioning. In task sequencing, the temperature after τ_1 finishes is $T(t_1)$. The temperature after τ_2 finishes is $T(t_1 + t_2)$. In task partitioning, the temperature after τ_{11} finishes is $T_p(t_{11})$. The temperature after τ_{21} finishes is $T_p(t_{11} + t_{21})$. The temperature after τ_{12} finishes is $T_p(t_1 + t_{12})$. The temperature after τ_{22} finishes is $T_p(t_1 + t_2)$.

A. Periodic tasks with common period

Consider a periodic set of N heterogenous tasks L, let P_i be the average power consumption during the execution time c_i of task τ_i. The goal is to find a sequence of these tasks using task partitioning to minimize the peak temperature. Due to the fact that all the tasks have the same common period, each task can be moved freely within the period. Furthermore, we need to analyze only one period, other periods will be the same as this one.

Methods that only reorder the sequence of tasks (such as TSA [12]) fail to further reduce the peak temperature when some "hot" tasks of long enough execution time can reach close to its steady-state temperature and this temperature is relatively independent of the initial temperature. We propose a task partitioning algorithm to reduce the temperature using preemption. The main idea of task partitioning algorithm is to partition the "hot" tasks into several subtasks and interleaving them with "cool" tasks to absorb the heat generated by "hot" subtasks. To partition "hot" tasks into more subtasks and generate enough "cool" subtasks to interleave with them, we divide the tasks into ***categories*** based on their power profile. The tasks in higher categories are partitioned into more subtasks. The subtasks in lower categories act as "cool" tasks and interleave with subtasks in higher categories. Details of the algorithm is shown in Algorithm 1.

Algorithm 1 Task Partitioning Algorithm (TPA)

1: Sort the tasks based on the power profile from coolest to hottest
2: Group the sorted tasks into k categories with equal number of tasks. These categories are numbered from 1 to k.
3: Partition tasks in category j, $2 \leq j \leq k$, into 2^{i-1} equal subtasks. Partition tasks in category 1 into 2 equal subtasks.
4: **for** $i = 1$ to $k - 2$ **do**
5: Interleave tasks of i^{th} category with tasks of $i+1^{th}$ category to form the new $i + 1^{th}$ category
6: **end for**// *Now only two categories are left. The first one is category k and the second one is a new category $k-1$ derived by combining category 1 through $k-1$.*
7: Insert the subtasks in new category $k-1$ into the intervals of tasks in category k.

First, we sort the tasks based on their power consumption from coolest to hottest. In the second step, we group the sorted tasks into k ***categories*** from category 1 to category k. Category 1 is the category of coolest n/k tasks, and category k is the category of hottest n/k tasks. The reason that we divide the tasks into different categories is that we need to partition "hotter" tasks into more subtasks to reduce the peak temperature. We also need enough "cooler" subtasks to separate the "hot" subtasks (see Figure 3). By having different categories of tasks, we can achieve both targets simultaneously. In this paper, we assume that when a task is partitioned into several small subtasks, the subtasks have the same average power consumption as the original task. We used Wattch [14] to compute average power of the subtasks. As expected, these numbers are comparable with the average power of the original tasks in our benchmark set.

In the next step, we partition tasks in category j, $2 \leq j \leq k$ into 2^{j-1} equal subtasks. The tasks in category 1 are partitioned into 2 equal subtasks. After recursively interleaving tasks in i^{th} category with tasks in $i+1^{th}$ category, there are only two categories left. The first one corresponds to category k and the second one is derived by combining category 1 through $k-1$. For the sake of convenience, we call this combined set as new category $k-1$. We now have $n/k \cdot 2^{k-1}$ tasks in category k, and $n/k \cdot 2^{k-1}$ intervals between these subtasks. We also have $n/k \cdot (2 + 2 + 4 + ... + 2^{k-2}) = n/k \cdot 2^{k-1}$ tasks in new category $k-1$. Therefore, we have enough tasks from category $k-1$ to interleave with the tasks in category k (see Figure 4). In the last step, we insert the tasks of new category $k-1$ into the intervals of category k (see Figure 5).

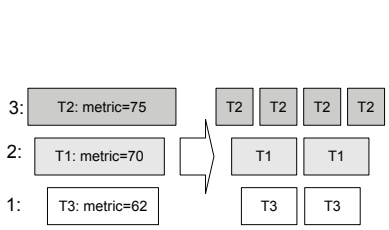

Fig. 3. Task Partitioning Algorithm (Step 1): Sort the tasks based on the power consumption, group the tasks into 3 categories. Task in category 3 is the hottest and task in category 1 is the coolest. Partition tasks into subtasks. Task in category 3 is partitioned into $2^{3-1} = 4$ equal subtasks. Task in category 2 is partitioned into $2^{2-1} = 2$ equal subtasks. Task in category 1 is partitioned into 2 equal subtasks.

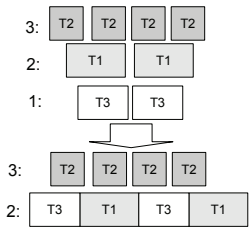

Fig. 4. Task Partitioning Algorithm (Step 2): Interleave subtasks of category 1 with subtasks of category 2 as the new category 2. Now we have enough subtasks from category 2 to interleave with the subtasks of category 3.

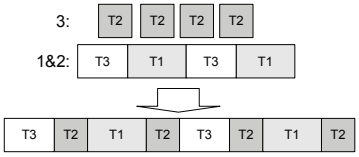

Fig. 5. Task Partitioning Algorithm (Step 3): We insert subtasks of new category 2 into intervals of subtasks of category 3 to get final sequence.

B. Periodic tasks with individual period

Consider a set of periodic N heterogeneous tasks in a set L where each task has its own period p_i. The arrival time a_i is equal to the start time of its period and the deadline d_i is equal to the end time of its period. Recall that the temperature profile of one hyper-period is identical to that of other hyper-periods. We only need to analyze the task instances within one hyper-period.

Theoretically, each periodic task corresponds to an infinite sequence of identical activities, called *instances*. The first instance of each periodic task arrives at time 0. Let P_i be the average power consumption during the execution time c_i of task τ_i. The goal is to find a sequence of these tasks using task partitioning to minimize the peak temperature.

The algorithm developed in the previous section does not apply to this scenario as the arrival and deadline constraints have to be carefully addressed. This additional constraint limits the task ordering that can be used. In this section, we develop a novel algorithm that integrates task partitioning technique into EDF scheduler[2]. We first use the EDF scheduler to schedule the tasks to get an initial

[2]Earliest-Deadline-First (EDF) [15] is a dynamic scheduling algorithm that schedules the tasks according to their absolute deadlines. Tasks with earlier deadlines will be executed at higher priorities.

sequence S_i. Based on the initial sequence, we can use Equation (3) to get the thermal profile of the task sequence. Using this thermal profile, let the task instance where peak temperature occurs be called *"hot" task instance*, denoted by τ_h.

The peak temperature is reduced by partitioning τ_h into several subtask instances interleaved by other "cool" task instances. Because the "hot" task instance τ_h cannot move before its arrival time or after its deadline, we only need to analyze the interval between the arrival time and deadline of τ_h, represented as a_{τ_h} and d_{τ_h}, called *hot interval*. All other task instances except the "hot" task instance in the hot interval are called *"cool" task instances*, denoted by τ_c. It is worth noting that this definition of "hot" and "cool" tasks is substantially different as compared to the previous section that considered periodic tasks with common period. A high level description of the EDF scheduling with our task partitioning is given in Algorithm 2.

Algorithm 2 EDF with task partitioning

```
1: Use EDF scheduler to get the initial schedule of these tasks
2: while loop for M times do
3:    Calculate the thermal profile of task sequence, find the "hot"
      task instance τ_h where peak temperature occurs.
4:    Partition the task instances whose execution period overlap
      with the arrival time or deadline of the "hot" task instance.
5:    In the hot interval, remove all the subparts of τ_h and calculate
      the available slack for each "cool" task instance.
6:    while there are parts of τ_h unassigned and some "cool" task
      instance has available slack do
7:       for each "cool" task instance τ_ci in the hot interval do
8:          if slack_i > 0 then
9:             Append one unit of τ_h into τ_ci and update the slack
               for all "cool" task instances
10:         end if
11:      end for
12:   end while
13:   If there is still some subparts of τ_h unassigned, scan the hot
      interval and assign them uniformly into the idle time.
14: end while
```

There are two major steps that are required for the Algorithm 2 and are as follows:

a) Partition the task instances whose execution period overlaps with the arrival time and/or deadline of the "hot" task instance: After finding the task instance τ_h, we can limit our analysis to the hot interval. It is likely that some task instances are across either a_{τ_h} or d_{τ_h}. We partition such task instances across these time lines using the following equation. If some task τ_k across the time line (either a_{τ_h} or d_{τ_h}) γ, that is, $s_{\tau_k} < \gamma < e_{\tau_k}$, where s_{τ_k} and e_{τ_k} are the start time and end time of τ_k, respectively. We have:

$$\tau_k|[s_{\tau_k}, e_{\tau_k}] \rightarrow \tau_k^{'}|[s_{\tau_k}, \gamma],\ \tau_k^{''}|[\gamma, e_{\tau_k}] \tag{5}$$

Where $\tau_k^{'}|[a_{\tau_k}, \gamma]$ means task $\tau_k^{'}$ starts executing at s_{τ_k} and will finish at γ.

The task instance τ_k is partitioned into at most 2 subtask instances, $\tau_k^{'}$ and $\tau_k^{''}$ based on the time line. We limit our further analysis to the subtask instance that is in the hot interval.

b) Slack allocation: EDF, in general, can break up a task into many subtasks to ensure the arrival and deadline constraints. In particular, "hot" task instance may have been decomposed into multiple subtasks. Thus there may exist more than one part of "hot" task instance τ_h in the hot interval.

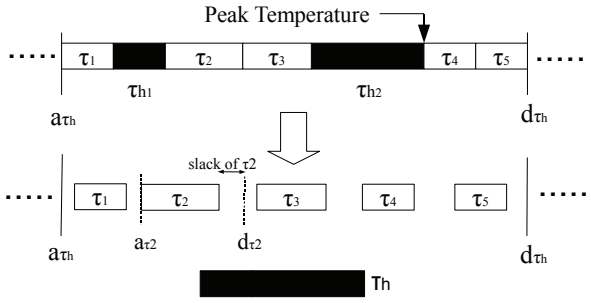

Fig. 6. First portion of the slack allocation step in the EDF based partitioning algorithm. Peak temperature occurs at task instance τ_h. a_{τ_h} and d_{τ_h} are the arrival time and deadline of τ_h, respectively. There are two parts of τ_h: τ_{h_1}, τ_{h_2}. Both τ_{h_1} and τ_{h_2} are removed from the sequence. Then other "cool" tasks instances (τ_{c1} - τ_{c5}) in the hot interval have more flexibility. Therefore, the slacks of these "cool" task instances can be calculated (For ease of presentation in this limited space, only τ_{c2}'s arrival time a_{τ_2}, deadline d_{τ_2} and slack are shown).

We first scan the hot interval and remove all the parts of the "hot" task instances from hot interval and combine them into one task (see Figure 6). Due to this step, other "cool" task instances in the hot interval will have more flexibility in time constraints. The extent of flexibility available under time constraints is quantified by the *slack*. Slack is defined by the difference between *Earliest Start Time* (EST) and *Latest Start Time* (LST). The definition of the EST, LST and slack of "cool" task instance τ_i can be computed as follows:

$$\begin{aligned} EST_i &= \max(a_{\tau_h}, a_{\tau_i}, EST_{pred_i} + c_{pred_i}) \\ LST_i &= \min(d_{\tau_h}, d_{\tau_i}, LST_{succ_i}) - c_i \\ slack_i &= LST_i - EST_i \end{aligned} \tag{6}$$

where $pred_i$ is the predecessor of τ_i defined by EDF schedule, $succ_i$ is the successor of τ_i defined by EDF schedule. Here, a_{τ_i} and d_{τ_i} are the arrival time and deadline of task instance τ_i, respectively, and c_i is the execution time of τ_i.

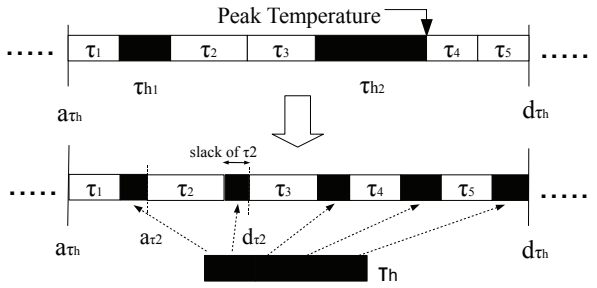

Fig. 7. Second portion of the slack allocation step in the EDF based partitioning algorithm. Calculating the slacks of all the tasks in the hot interval. For each task τ_{ci}, append one *unit* of τ_h into τ_{ci} at a time and update the slacks. When no slacks left and there are still some parts of τ_h unassigned, scan the hot interval and assign them uniformly into the idle time.

After the available slacks have been calculated for all the "cool" task instances, we insert the "hot" task instance back into these slacks as uniformly as possible. This is done by appending one *unit* slack of τ_h to each "cool" task instance at a time (one *unit* slack is a small constant representing a small period of time). If there is still some subparts of τ_h unassigned, there must exist some idle time that no "cool" task instance can have slack on it. The remaining part of the τ_h is uniformly decomposed into these idle times (see

Figure 7).

The above process corresponds to a single iteration of the algorithm. This process is applied iteratively for several iterations. In each iteration, a potentially new "hot" task instance is chosen based on the thermal profile. The number of iterations or loops that should be iterated over can be fixed or chosen based on the level of improvement achieved. For our experiments, we found that 10-15 iterations are sufficient for deriving most of the benefits.

IV. Experimental Results

We used a platform that is based on ARM Cortex A8 [16]: 2-width in-order issue, 32KB instruction and data caches for evaluating our algorithms. The clock speed was set to 1.5GHz. Using default thermal configurations in HotSpot [17] and the floorplan and silicon area of ARM Cortex A8, the thermal resistance and capacitance can be computed as $1.83^oC/Watt$ and $0.112J/^oC$, respectively [12]. The ambient temperature was set at 45.15^oC. We used the architecture-level power simulator Wattch [14] to obtain the power consumption of tasks.

A. Tasks with Common Period

For tasks with common period, synthetic tasks are generated to find the profitable number of categories to achieve a lower peak temperature and real benchmarks are generated to compare the performance between task sequencing algorithm (denoted by TSA [12]) and our task partitioning algorithm (denoted by TPA). For tasks with individual period, real benchmarks are generated.

1) Synthetic Tasks: Tasks were generated to compare the thermal reduction achieved by our task partitioning algorithm with different number of categories. The numbers of clock cycles of tasks are uniformly distributed in $[1.5\times10^8, 1.45\times10^9]$, the power consumption of these jobs are uniformly distributed in $[5, 25]$ Watt. The numbers of jobs tested are $32, 64, 128, 192, 256$. The numbers of categories in task partitioning are $2, 3, 4, 5$.

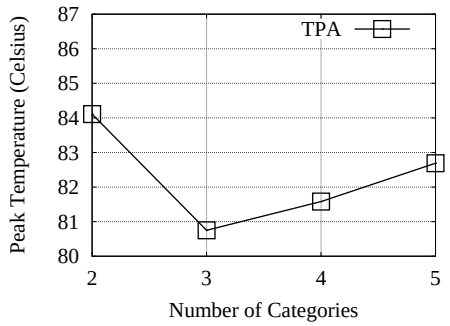

Fig. 8. Peak temperature comparison between different number of categories.

Figure 8 shows the peak temperature comparison using different number of categories for the task partitioning algorithm. The number of tasks used in this scenario is 64. Recall from the task partitioning algorithm, different number of categories will lead to different number of partitioning within each categories. It is necessary to find a profitable number of categories to achieve a lower peak temperature. The results show that 3 categories are enough to achieve most of the benefits for reducing the peak temperature. In fact using larger number of categories may result in slightly higher temperature. This is because when the number of categories is large, the number of tasks in the highest category is fewer. Some tasks originally in the highest category will be pushed into the second highest category. These tasks, who are already very hot, are treated as "cooler" tasks to interleave with tasks in the highest category. However, these tasks cannot absorb enough heat from the "hot" tasks. This can lead to the algorithm effectively not able to reduce the temperature significantly. Our experiments suggest that 3 categories should be ideal for most practical scenarios.

2) Real Benchmarks: We use Mibench [18] and Mediabench [19] to form four sets of the benchmark tasks. The characteristics of these benchmarks are shown in Table I.

TABLE I
THE FOUR SETS OF BENCHMARK TASKS WITH COMMON PERIOD

set1	patricia, adpcm, rijndael, susan, crc, FFT, dijkstra, epic
set2	patricia, djpeg, adpcm, sha, FFT, rijndael, susan, rijndael
set3	sha, djpeg, FFT, rijndael, dijkstra, epic, rijndael, susan
set4	rijndael, dijkstra, FFT, gsm, sha, patricia, pegwit, djpeg

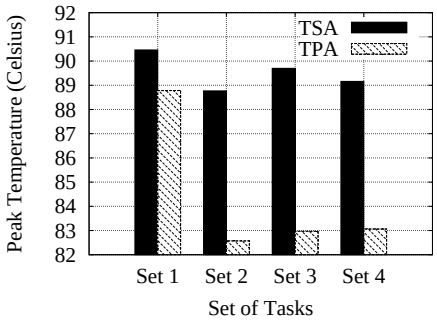

Fig. 9. Peak temperature comparison between task sequencing algorithm and task partitioning algorithm on real tasks. Number of categories is 3.

Figure 9 shows the peak temperature comparison between task sequencing algorithm and our task partitioning algorithm on real tasks for 3 categories. The task partitioning algorithm reduces the peak temperature by as much as 5.8^oC compared with task sequencing algorithm. Given the above results, a choice of 3 categories should generally provide a good tradeoff between low context switching overhead and high reduction in peak temperature as discussed in Section IV-C.

B. Tasks with Individual Period

For tasks with individual period, we also use Mibench [18] and Mediabench [19] to form four sets of the benchmark tasks. These benchmarks are shown in Table II. Each periodic task has individual periods. We set the deadline of all tasks to be the end of its own period. Also, we assume that arrival time of the first instance of all tasks is 0.

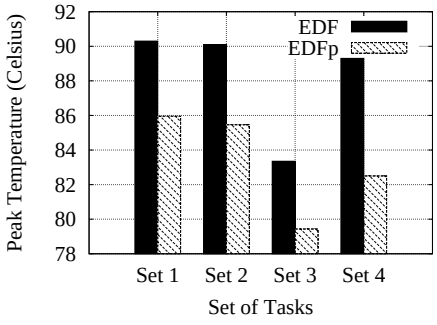

Fig. 10. Peak temperature comparison between EDF and EDFp ($M = 15$).

We compare our approach with EDF that does not directly address thermal issues. Figure 10 shows the peak temperature comparison

TABLE II
THE FOUR SETS OF BENCHMARK TASKS WITH INDIVIDUAL PERIOD

Set Number	Benchmark	Execution Time (Clock Cycles)	Period (Clock Cycles)
set1	patricia	1.32×10^8	7.2×10^8
	adpcm	1.81×10^7	1.35×10^8
	susan	1.48×10^7	2.66×10^8
	crc	2.13×10^8	9.73×10^8
	dijkstra	2.9×10^7	1.74×10^8
set2	pegwit	2.12×10^7	2.72×10^7
	gsm	6.35×10^6	9.52×10^7
	epic	3.21×10^7	1.44×10^8
	crc	2.13×10^8	9.73×10^8
	patricia	1.32×10^8	7.2×10^8
set3	gsm	6.35×10^6	9.52×10^7
	patricia	1.32×10^8	7.2×10^8
	susan	1.48×10^7	2.66×10^8
	djpeg	1.57×10^7	9.42×10^7
	adpcm	1.81×10^7	1.35×10^8
set4	rijndael	4.5×10^7	3.6×10^8
	susan	1.48×10^7	2.66×10^8
	FFT	1.54×10^8	1.1×10^9
	sha	4.8×10^7	2.7×10^8
	adpcm	1.81×10^7	1.35×10^8

between EDF and our approach, called EDFp. The experimental results show that the EDFp outperforms EDF by as much as 6^oC.

C. Context Switching Overhead

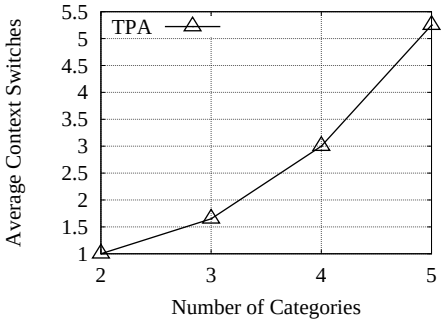

Fig. 11. Average number of context switches per task of task partitioning algorithm for various number of categories.

We first analyze the context switching overhead introduced by partitioning the tasks with common period. Figure 11 shows the number of context switch per task for task partitioning algorithm using variable number of categories. For task partitioning algorithm with 3 categories (which is shown most profitable in Figure 8), the number of context switches is about 1.8 per task, which is tolerable in many practical scenarios [3].

Second, we analyze the context switching overhead for tasks with individual periods. Figure 12 shows the average number of context switches per task between EDF and EDFp for various sets of tasks. The overhead for EDFp (less than 2 context switches per task) is tolerable for most practical scenarios.

V. CONCLUSION

Both power and heat density of on-chip systems are increasing exponentially with Moore's Law. High temperature negatively affects reliability as well the costs of cooling and packaging. In this paper, we propose task partitioning as an effective way to reduce the peak temperature in the embedded systems. We developed novel algorithms that address two broad scenarios: (1) a set of periodic tasks with common period and (2) a set of periodic tasks with individual period. Experimental results show that our first algorithm outperforms the task sequencing algorithm [12] by reducing the peak temperature by as much as 6^oC. For task sets with individual period, EDF scheduling with task partitioning can lower the peak temperature, as compared to simple EDF scheduling, by as much as 6^oC. Our analysis indicates that the number of additional context switches (overhead) is less than 2 per task, which is tolerable in many practical scenarios. These results are promising and clearly demonstrate that task partitioning is an effective way to reduce the peak temperature in embedded systems.

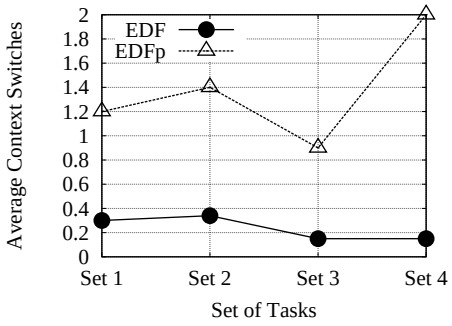

Fig. 12. Average number of context switches per task comparison between EDF and EDFp for various sets of tasks.

[3]Context switch time on ARM cpu can be less than $10us$ [20]

REFERENCES

[1] K. Skadron et al., "Temperature-aware computer systems: Opportunities and challenges," *IEEE Micro*, vol. 23, no. 6, pp. 52–61, 2003.
[2] S. Gunther et al., "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, 5(1), pp. 1–9, 2001.
[3] S. Gunther et al., "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, vol. 1, pp. 1–9, 2001.
[4] *Failure mechanisms and models for semiconductor devices*, jedec.org.
[5] K. Skadron et al., "Temperature-aware microarchitecture: Modeling and implementation," *TACO*, vol. 1, no. 1, pp. 94–125, 2004.
[6] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *HPCA*, 2001, p. 0171.
[7] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors," in *ICCAD*, 2008, pp. 537–542.
[8] M. Kadin and S. Reda, "Frequency planning for multi-core processors under thermal constraints," *ISLPED*, 2008, pp. 213–216.
[9] S. Murali et al., "Temperature control of high-performance multi-core platforms using convex optimization," in *DATE*, 2008, pp. 110–115.
[10] T. Ebi et al., "Tape: thermal-aware agent-based power economy for multi/many-core architectures," *ICCAD*, 2009, pp. 302–309.
[11] R. Ayoub and T. Rosing, "Predict and act: dynamic thermal management for multi-core processors," in *ISLPED*, 2009, pp. 99–104.
[12] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *ICCAD*, 2008, pp. 618–623.
[13] J. Choi et al., "Thermal-aware task scheduling at the system software level," in *ISLPED*, 2007, pp. 213–218.
[14] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *Computer Architecture News*, vol. 28, no. 2, p. 94, 2000.
[15] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *SOSP*, 2001, p. 102.
[16] ARM, www.arm.com/products/processors/cortex-a/cortex-a8.php.
[17] University of Virginia, http://lava.cs.virginia.edu/HotSpot/.
[18] M. Guthaus et al., "Mibench: A free, commercially representative embedded benchmark suite," in *WWC*, 2001, pp. 3–14.
[19] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *MICRO*, 1997, pp. 330–335.
[20] SEGGER, http://www.segger.com/cms/context-switching-time.html.