Efficient Task Partitioning and Scheduling for Thermal Management in Multicore Processors

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*—Power and heat density of integrated circuits (ICs) are rising exponentially over the years. The overheating of ICs leads to higher cost of cooling and packaging as well as reliability concerns and shorter lifetime. While existing task-partitioning based approaches are promising for reducing peak temperature in uniprocessor systems, there are no previous efforts in exploring temperature-aware task partitioning in multicore architectures. In this paper, we propose a task partitioning and scheduling algorithm to reduce the hot-spot in multicore embedded systems running a set of independent tasks. Experimental results using real benchmarks show that our approach is able to reduce the peak temperature by as much as 4.52^oC compared to the state-of-the-art thermal-aware task scheduling algorithm PDTM [22] while requires 31% less time to finish all the tasks.**

I. Introduction

Thermal management is widely acknowledged as a major concern in designing integrated circuits (ICs). Temperature increase directly impacts the performance and reliability of ICs. Studies [2] have shown that 10-15^oC increase in peak temperature can reduce the lifetime of the chip by half. Moreover, increase in power consumption increases the cost of cooling and packaging [1].

Existing power-aware techniques do not address the temperature issues in embedded systems. The main reason is that the power distribution of multicore processors is not uniform. Localized heating rises more rapidly than the whole chip, leading to nonuniform temperature distribution on the chip with localized high-temperature hot spots and spatial gradients [3]. Traditional methods to control 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 thermal impact on various on-chip parameters and incorporate effects of non-uniform thermal profiles during integrated circuit design process. However, such technologies are unable to deal with a variety of runtime situations.

In this paper, our focus is 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 modeled using RC model [3]. If the average power of a design 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:

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

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. For a given design 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. Existing research efforts tried to reduce these three factors that led to the following research directions to reduce the on-chip temperature.

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 [4]–[9]. Huang et al. [13] present a trigger technique that throttle the processor by either reducing the voltage and frequency or toggling other units when the temperature is higher than some trigger threshold. A few proactive thermal management techniques have also been suggested to prevent overheating problems in chip [9]. All of the above techniques utilize DVFS to reduce the power consumption by lowering the voltage and frequency of the processors. Unfortunately, these techniques have limited impact on the temperature if the available slack is small (deadline constraints are strict). Moreover, these approaches will extend execution time of tasks, which may lead to longer time to finish all the tasks.

Thread migration or coolest-first techniques have been used to lower the initial temperature of each task, thereby, lowering the final temperature [14], [15], [22]. Temperature aware task sequencing algorithm, which reduces the initial temperature [10], 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" tasks[1] are long. The algorithm to defer execution of hot tasks [11] fails to reduce temperature similarly. This is because when the execution time of a "hot" task is long, it can lead to a high steady-state temperature irrespective of the initial temperature.

This work was supported in part by NSF grants CNS-1441667 and CCF-1218629 and SRC grant 2014-TS-2554.

[1]A "hot" task is a task with higher average power consumption, and a "cool" task is a task with lower average power consumption.

Task Partitioning Algorithm (TPA) is proposed as an algorithm to reduce the peak temperature by reducing the "hot" tasks' execution time on single core processor [12]. This approach can reduce peak temperature of processors without slowing down the total execution time of tasks. However, there are many challenges in applying task partitioning for multicore architectures. On single-core processors, only temporal temperature distribution is considered. On multicore processors, we need to also consider the spatial temperature distribution between cores to reach roughly balanced temperature throughout the chip. It is challenging to find an appropriate partitioning method to simultaneously balance temporal and spatial temperature distribution together.

In this paper, we propose a heuristic algorithm that utilizing Task Partitioning to reduce the peak temperature on multicore processors. Our algorithm contains both intra-core scheduling and inter-core scheduling steps. Intra-core scheduling balances temporal temperature distribution within cores by partitioning "hot" tasks into multiple subtasks and interleave these subtasks with "cool" tasks. Inter-core scheduling balances the spatial temperature distribution by aligning the task sequences between neighboring cores. Experimental results show that our algorithm outperforms the existing state-of-the-art approaches PDTM by reducing the peak temperature by as much as 4.52^oC while requires 31% less time to finish all the tasks.

The rest of the paper is organized as follows. Section II proposes our task partitioning and scheduling algorithm to reduce peak temperature in multicore processors. Section III compares this algorithm using real benchmarks with state-of-art thermal-aware task schedule algorithm PDTM. Section IV concludes the paper.

II. Task Partitioning and Scheduling (TPS)

We first give an illustrative example showing that TPS can reduce the peak temperature on a multicore processor running independent heterogeneous tasks. Figure 1 shows two scheduling scenarios in a dual-core processor. The first scheduling result is using Min-Min algorithm [17]. The other one is based on our TPS algorithm. We can see that TPS breaks long "hot" tasks into subtasks and interleave them with "cool" subtasks to reach temporal heat balance. It also schedules the "hot" and "cool" subtasks between cores to reach spatial heat balance.

Figure 2 shows the transient peak temperature comparison between Min-Min algorithm and TPS algorithm on a dual-core processor. We can see that our TPS algorithm can achieve lower peak temperature.

We then give the detailed explanation of TPS algorithm. Consider a set of independent heterogeneous tasks (with different thermal profiles), with the execution time of these tasks given by $e_1, e_2, ..., e_N$. Let P_i be the average power consumption during the execution time of e_i. The goal is to find a sequence of these tasks to a homogeneous multicore processor using task partitioning to minimize the peak temperature. Our algorithm mainly consists of the following four steps. The remainder of this section describes these steps in detail.

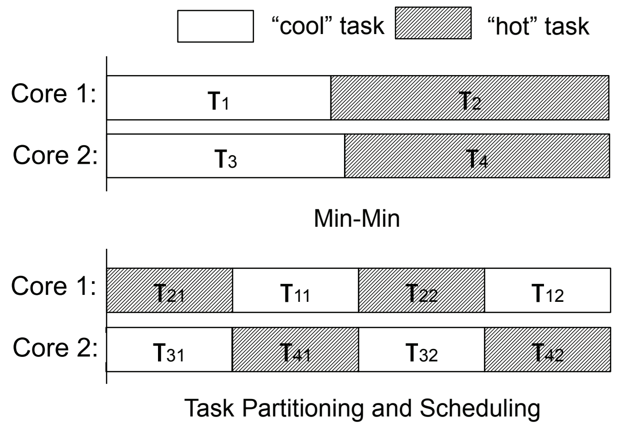

Fig. 1. The task scheduling results of Min-Min algorithm and TPS algorithm on a dual-core processor. There are 4 independent tasks to be scheduled. Task τ_1 and τ_3 are "cool" tasks. Task τ_2 and τ_4 are "hot" tasks. TPS algorithm breaks each task into two subtasks as follows: task τ_1 (τ_{11} and τ_{12}), task τ_2 (τ_{21} and τ_{22}), task τ_3 (τ_{31} and τ_2), nd task τ_4 (τ_{41} and τ_{42}).

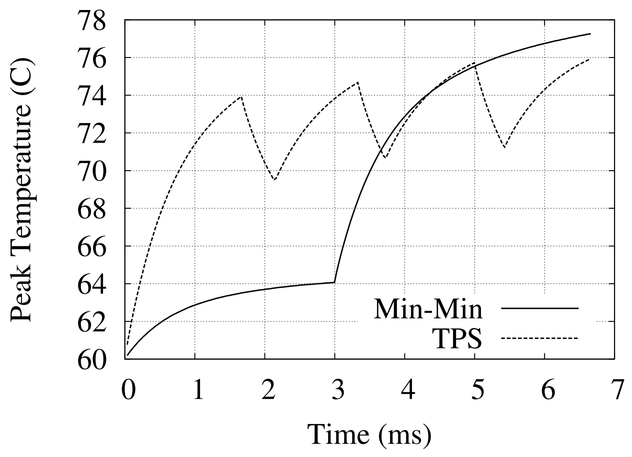

Fig. 2. The transient peak temperature comparison between Min-Min algorithm and TPS algorithm on a dual-core processor.

1) In order to facilitate scheduling of the "hot" and "cool" tasks between different neighboring cores, we need to first normalize the execution time of each task.
2) Use task assignment algorithm to assign tasks onto multicore processors.
3) Use task partitioning algorithm within each core to partition the long "hot" tasks into short "hot" subtasks and interleave them with "cool" tasks to absorb the heat generated by "hot" subtasks.
4) Schedule the execution of subtasks between cores to make sure that when one core is executing a "hot" task, all the neighboring cores are executing "cool tasks.

Normalize the Execution Time of Each Task: First, we normalize the execution time of all tasks. It is difficult to schedule "hot" and "cool" tasks within one core as well as aligning them between cores in the same time. We normalize the execution time of each tasks as integer multiple of δ, which will be used the minimum scheduling unit in next steps. DVFS technique is used to normalize the task execution by carefully adjust the core frequency. Let e_i be the execution time of task τ_i. It is extended to the smallest integer multiple of δ larger than e_i. Let E_i be the normalized execution time of task τ_i in terms of δ, then, $E_i = \lceil \frac{e_i}{\delta} \rceil$. Figure 3 shows an example of normalizing execution time of two tasks.

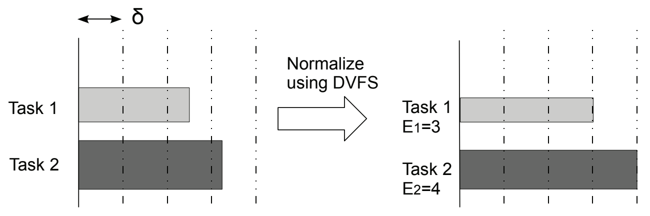

Fig. 3. Use DVFS to normalize the execution time of tasks to the smallest integer multiple of δ. Task 1 is normalized to 3 times of δ, $E_1 = 3$. Task 2 is normalized to 4 times of δ, $E_2 = 4$.

Task Assignment: We use the Min-Min algorithm [17] as our task assignment algorithm. Min-Min algorithm assigns these tasks to cores and it is expected to generate small makespan [18]. Makespan M is the total execution time of a set of tasks. Makespan is defined as: $M = \max_j\{\sum_{\tau_i \in \text{core j}} E_i\}$.

In step 1, we normalize the time into multiple time slots, each of them is as long as δ. We use $T_{k,j}$ as the temperature of $core_j$ at time slot k. Matrix A, vector B and C are based on the thermal characteristic of the chip [8]. Using DVFS to extend the execution time will lowering the power consumption of the tasks. Let P'_i be the new power consumption of task τ_i after normalization. We use integer $l_{k,i,j}$ as the label to show task schedule results. $l_{k,i,j} = 1$ means core j will execute task i on time slot k, otherwise $l_{k,i,j} = 0$. T_{init} is the initial temperature. Then the problem can be formulated as follows:

$$\text{Minimize:} \quad T_{peak} \tag{2}$$
$$\text{such that: } T_{peak} \geq T_{k,j}, \forall k, j \tag{3}$$
$$T_{k+1,j} = T_{k,j} + \sum_{\forall n \in Ad_j} A_{n,j}(T_{k,n} - T_{k,j}) + B_j P_{k,j} + C_j(T_A - T_{k,j}), \forall k, j \tag{4}$$
$$T_{0,j} = T_{init}, \forall j \tag{5}$$
$$P_{k,j} = P'_i, \quad \text{if} \quad l_{k,i,j} = 1, \forall k, j \tag{6}$$
$$\sum_k l_{k,i,j} = E_i, \forall \tau_i \in core_j \tag{7}$$
$$\sum_i l_{k,i,j} = 1, \forall k, j \tag{8}$$
$$l_{k,i,j} \in \{0, 1\} \tag{9}$$

Equation (4) calculates the temperature on each time slot on each core. Equation (6) use the power consumption of the task scheduled on time slot k of core j as the power consumption of $P_{k,j}$. Equation (7) is the constraint that task τ_i need E_i number of time slots. Equation (8) is the constraint that only one task can be scheduled per time slot per core. This problem is an integer programming problem and not possible to generate optimal results in a reasonable time. Therefore we propose a heuristic algorithm in next two steps to solve it.

Intra-core Scheduling: Intra-core scheduling reduces the peak temperature in a temporal way. In this step, we only focus on the scheduling of task sequences within each core. Please note that after task assignment, some cores might have smaller total execution time than other cores. These cores may become idle when waiting other cores to finish their tasks. We will utilize these idle times to further reduce the temperature. The idle time of each core j will be $I_j = M - \sum_{\tau_i \in \text{core j}} E_i$. Then we treat the idle time as a task τ_{I_j} with power as core idle power and execution time as I_j. Task τ_{I_j} will be considered as a regular task and be scheduled with other tasks together.

Algorithm 1 Intra-core Scheduling

```
1: Sort the tasks on their average power consumption in ascending order within each core. Assume that $N_j$ is the number of tasks assigned to core $j$.
2: Compute the summation of $E_i$ of all tasks $\tau_i$ on assigned core $j$, denoted by $E_{sum_j}$.
3: Find the task $\tau_{m_j}$ such that $\sum_{i=1}^{m_j-1} E_i < E_{sum_j}/2$ and $\sum_{i=1}^{m_j} E_i \geq E_{sum_j}/2$. Define tasks from 1 to $m_j$ as "cool" tasks, tasks from $m_j + 1$ to $N_j$ as "hot" tasks.
4: Partition all the tasks into subtasks with execution time of $\delta$. Then task $\tau_i$ is partitioned into $E_i$ subtasks.
5: By recursively pairing the two subtasks from two end of the task list (one is "cool" subtask, one is "hot" subtask) while maintaining the order of subtasks from the same parent task, we can efficiently interleave the "cool" and "hot" subtasks.
```

Algorithm 1 shows our intra-core scheduling procedure. First, we sort the tasks based on their average power consumption in ascending order within each core (step 1). Suppose there are N_j tasks assigned to core j. For each core j, number the tasks on core j from 1 to N_j. Then, we need to define which tasks are "hot" tasks and which are "cool". Step 2-3 make sure there are enough "cool" subtasks to separate "hot" subtasks. Next, we partition all the tasks into subtasks with execution time δ (step 4). Recursively pairing the two subtasks from two end of the task list, "cool" and "hot" subtasks can be effectively interleaved (step 5). 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 [19] to compute average power of the subtasks. These numbers are comparable with the average power of the original tasks. An example of intra-core scheduling on 4 tasks is shown in Figure 4.

Inter-core Scheduling: In intra-core scheduling, we only focus on the scheduling within each core. In this step, we need to consider the mutual influence between cores. Inter-score scheduling works as follows: fixing the subtask sequence on one core, starting with a "cool" subtask, followed by a "hot" subtask, followed by a "cool" subtask, etc. All the neighboring cores was set with an opposite sequence. And continue to

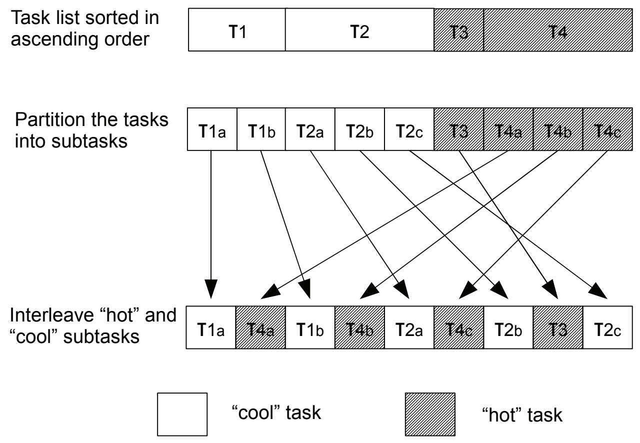
Task list sorted in ascending order
T1
T2
T3
T4
Partition the tasks into subtasks
T1a
T1b
T2a
T2b
T2c
T3
T4a
T4b
T4c
Interleave "hot" and "cool" subtasks
T1a
T4a
T1b
T4b
T2a
T4c
T2b
T3
T2c
"cool" task
"hot" task

Fig. 4. The process of intra-core scheduling. 4 tasks are sorted in ascending order on their average power consumption. Task τ_1 and τ_2 are "cool" tasks. Task τ_3 and τ_4 are "hot" tasks. Task τ_1 is partitioned into subtasks τ_{1_a} and τ_{1_b}, Task τ_1 into subtasks τ_{2_a}, τ_{2_b} and τ_{2_c}, etc. Pairing "cool" and "hot" subtasks while maintaining the order between subtasks from same parent task.

set task sequence on other cores following this rule until the task sequence on all the cores were settled. Since each subtask has exactly the same execution time δ, they will align automatically with all "hot" subtasks surrounded by "cool" subtasks. Algorithm 2 shows our inter-core schedule procedure in detailed steps. Our inter-core scheduling algorithm is developed based on Breadth First Search (BFS) to set the neighbor cores with opposite task sequence. We first fix $core_0$'s task sequence, then we set the task sequences of $core_0$'s neighbor cores to be opposite of $core_0$. We use a queue to record which core need to visit and use visited flags to indicate whether the core is visited or not. When the queue is empty, it means we visited all the cores and set their task sequences accordingly. An example of final state of task sequence on a 4-core processor is shown in Figure 5.

Algorithm 2 Inter-core Scheduling

```
1: Initialize a queue $Q$.
2: for each core $j$ do
3:     $visited[j] := false$
4: end for
5: Set core 0's task sequence as: $\{CHCHCH...\}$
   // $C$ denotes "cool" subtask, $H$ denotes "hot" subtask
6: ENQUEUE(Q, core 0) // Insert core 0 in Q
7: while $Q$ is not empty do
8:     $\hat{core}$ =DEQUEUE(Q) // Delete the front element of Q
9:     for each core $j \in$ {neighbor of $\hat{core}$} do
10:        if $visited[j] = false$ then
11:            Set core $j$'s task sequence opposite to sequence of $\hat{core}$
12:            $visited[j] = true$
13:            ENQUEUE(Q, core $j$)
14:        end if
15:    end for
16: end while
```

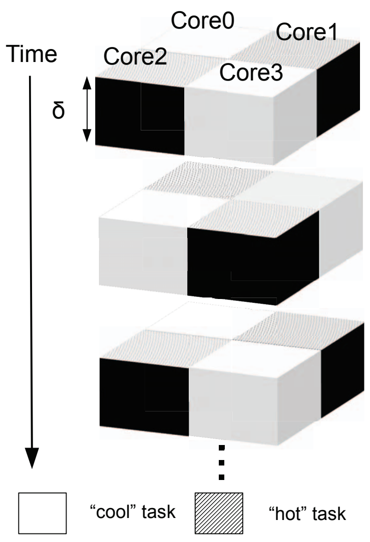

Fig. 5. Inter-core scheduling. There are 4 cores. Core 0 has neighboring Core 1 and Core 2. Therefore, Core 1 and core 2 have opposite "hot" and "cool" subtask sequence to core 0. Core 3 is neighbor to core 1 and core 2 and it has the same sequence as core 0.

III. Experimental Results

We used a platform based on ARM Cortex A9 [20]: 2-width out-of-order issue, 32KB instruction and data caches for evaluating our algorithms. The clock speed was set to 1.5GHz. Default thermal configurations in HotSpot [21] and the floorplan of ARM Cortex A9 are used. The ambient temperature was set at 45.15^oC. We used the architecture-level power simulator Wattch [19] to obtain the power consumption of tasks. We have conducted experiments using 4-core, 8-core as well 16-core architectures. In this section, we present results for 16-core scenario. The distribution of the 16 cores is a 4×4 matrix. To measure the proper temperature when processors runs tasks for long time. We set the average steady-state temperature of the task sequence as the initial temperature and run the tasks for 3 loops. We take the temperature in the final loop as measured temperature. We compared the peak temperature and makespan (the time taken to finish all tasks) between three algorithms. The algorithms compared are as follows:

- Min-Min: Use only Min-Min algorithm to assign tasks to the multicore processor.
- PDTM: Predictive Dynamic Thermal Management, a state-of-art dynamic temperature-aware task scheduling algorithm. PDTM can achieve lower temperature by migrating process from high-temperature cores to potentially coolest cores in future [22].
- TPS: We use four TPS algorithms based on different values of δ. They are TPS-1($\delta = 0.33ms$), TPS-2($\delta = 0.66ms$), TPS-3($\delta = 1.32ms$) and TPS-4($\delta = 2.64ms$).

We use Mibench [23] and Mediabench [24] to form four sets of benchmark tasks, which are shown in Table I. Each benchmark set has 30 benchmark tasks.

TABLE I
THE FOUR SETS OF BENCHMARK TASKS

Real Benchmark Tasks	
Set 1	dijkstra, patricia, adpcm, crc32, fft gsm, ghostscript, cjpeg, blowfish, stringsearch, sha
Set 2	epic, unepic, rasta, epegwit, ifft dpegwit, ghostscript, dadpcm, ungsm blowfish, susan, basicmath
Set 3	dijkstra, adpcm, crc32, fft, gsm, rasta ghostscript, jpeg, patricia, rijndael blowfish, stringsearch, sha, epic, pegwit
Set 4	stringsearch, rasta, unepic, dadpcm, ifft susan, c, ghostscript, dpegwit, blowfish crc32, dijkstra, djpeg, sha

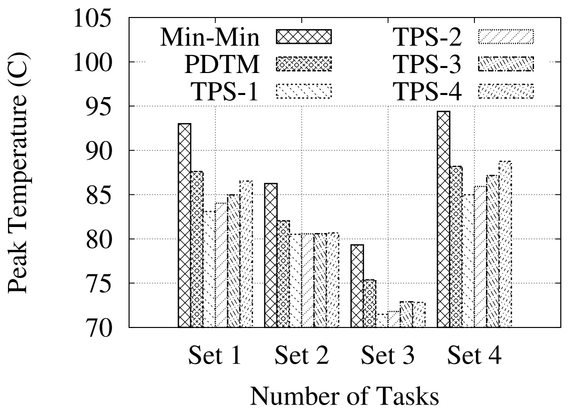

Fig. 6. Peak temperature comparison between Min-Min, PDTM and our TPS algorithms for real benchmarks on a 16-core processor.

Figure 6 shows the peak temperature comparison between Min-Min, PDTM and our TPS algorithms on a 16-core processor. We can see that TPS achieves significantly lower peak temperature compared with PDTM algorithms, reducing the peak temperature by up to 4.52^oC.

Figure 7 shows the makespan comparison between Min-Min, PDTM and our TPS algorithms on a 16-core processor. We can see the time to finish the tasks (makespan) scheduled by TPS algorithm is 31% less compared with PDTM. So TPS can significantly reduce the peak temperature on multicore processors while largely improved the time needed to execute the tasks.

IV. Conclusion

Increasing power and heat density requires efficient thermal management techniques in embedded systems. High temperature negatively affects reliability as well the costs of cooling and packaging. In this paper, we developed an efficient task partitioning and scheduling technique to reduce the peak temperature in multicore processors. Our experimental results can reduce the peak temperature up to 4.52^oC compared with PDTM using real benchmarks. In terms of speed, our approach requires 31% less time to finish all the tasks.

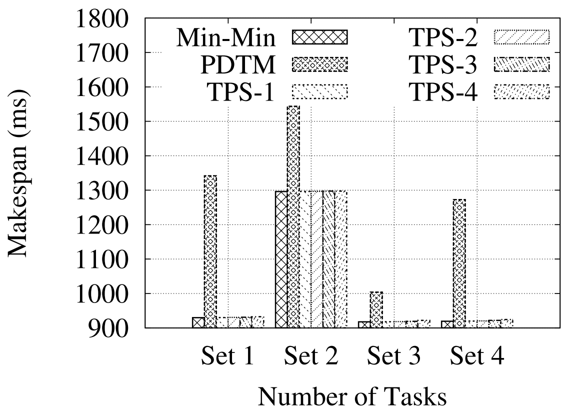

Fig. 7. Makespan comparison between Min-Min, PDTM and our TPS algorithms for real benchmarks on a 16-core processor.

References

[1] S. Gunther et al., "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, 5(1), 2001.
[2] *Failure mechanisms and models for semiconductor devices*, jedec.org.
[3] K. Skadron et al., "Temperature-aware microarchitecture: Modeling and implementation," *TACO*, 1(1), pp. 94–125, 2004.
[4] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *HPCA*, 2001.
[5] Y. Liu et al., "Thermal vs energy optimization for dvfs-enabled processors in embedded systems," in *ISQED*, 2007.
[6] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors," in *ICCAD*, 2008.
[7] M. Kadin and S. Reda, "Frequency planning for multi-core processors under thermal constraints," in *ISLPED*, 2008.
[8] S. Murali et al., "Temperature control of high-performance multi-core platforms using convex optimization," in *DATE*, 2008.
[9] T. Ebi et al., "Tape: thermal-aware agent-based power economy for multi/many-core architectures," in *ICCAD*, 2009.
[10] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *ICCAD*, 2008.
[11] J. Choi et al., "Thermal-aware task scheduling at the system software level," in *ISLPED*, 2007.
[12] Z. Wang et al., "Temperature-aware task partitioning for real-time scheduling in embedded systems," in *VLSI Design*, 2012.
[13] M. Huang et al., "A framework for dynamic energy efficiency and temperature management," in *MICRO*, 2000.
[14] K. Stavrou and P. Trancoso, "Thermal-aware scheduling for future chip multiprocessors," *EURASIP JES*, vol. 2007, no. 1, p. 40, 2007.
[15] Y. Xie and W. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip design," *JVSP*, 2006.
[16] M. Gomaa et al., "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," *ASPLOS*, 2004.
[17] R. Freund et al., "Scheduling resources in multi-user, heterogeneous, computing environments with smartnet," HCW 1998.
[18] M. Maheswaran et al., "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," HCW 1999.
[19] D. Brooks et al., "Wattch: a framework for architectural-level power analysis and optimizations," *ISCA*, 2000.
[20] ARM, http://www.arm.com/products/processors/cortex-a/cortex-a8.php.
[21] University of Virginia, http://lava.cs.virginia.edu/HotSpot/.
[22] I. Yeo et al., "Predictive dynamic thermal management for multicore systems," DAC 2008.
[23] M. Guthaus et al., "Mibench: A free, commercially representative embedded benchmark suite," in *WWC-4*, 2001.
[24] C. Lee et al., "Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *MICRO* 1997.