ThresHot: An Aggressive Task Scheduling Approach in CMP Thermal Design

Lin Li, Xiuyi Zhou, Jun Yang
Department of Electrical and Computer Engineering
University of Pittsburgh, Pittsburgh, PA 15261
Email: lil53, xiz44, juy9@pitt.edu

Victor Puchkarev
School of Arts & Sciences
University of Pittsburgh, Pittsburgh, PA 15260
Email: vgp2@pitt.edu

***Abstract*—In CMP (chip-level multiprocessor) chips, due to both the temporal and spatial temperature variations, there exist considerable potential to reduce the thermal emergencies and to improve the performance of jobs by designing smart thermal scheduling algorithm. However, scheduling jobs on multiple cores in CMP introduces the side-effect of thermal cycling, which harms the chip lifetime. In this paper we propose an aggressive job scheduling algorithm, ThresHot, to reduce the thermal emergencies, to improve the throughput and reliability by reducing the thermal cycling effect. For the SPEC2K benchmarks, the ThresHot saves 13% execution time in penalized period compared to the Baseline and 6% to the Balancing algorithm. Also, the thermal cycling effect is minimal in all existing schedule algorithms.**

***Index Terms*—Task Scheduling, Thermal Design, CMP.**

I. Introduction

With the semiconductor technology advancing to the nanometer regime, the immense transistor density leads to increasing intractable power density. With cooling budget limit, the increasing power injection will overheat the chip package and degrade the chip reliability. Moreover, running multiple heterogeneous jobs/threads on the CMP generate multiple hotspots, introducing more complexities for the chip thermal design. Unless proper power and thermal managements are provided, the chip will suffer from boosting power density, resulting deteriorated reliability, higher cooling cost, and poor performance.

To alleviate the chip from overheating working conditions, DTM (Dynamic Thermal Management) is used to throttle the power injection and to control the peak temperature of the chip. Although quite effective in reducing the thermal emergencies and peak temperatures, the DTM actions introduce the performance degradation, including: overhead of triggering DTMs and DTM frequency scaling. The situation is worse if DTM is applied at the chip level. In [1], applying DTM independently at core level yield less performance degradation. However, more complex hardware is required to support core level DTM.

Within power budget limit, the OS assisted thermal-aware job scheduling approach is effective and flexible to mitigate the thermal burden of the chip, especially in CMP. On a periodical basis, scheduling a hot job on an available cool core substantially reduce the peak temperature than assigned on a hot core. The scheduling problem is to determine how to assign N jobs on the N cores on consecutive scheduling points. By exploiting both the temporal and spatial temperature variations in CMP, the scheduling algorithm may smartly assigning hot jobs on cool cores to reduce the thermal emergencies. The temporal variation is due to the fluctuating power of jobs in different execution phases, providing temperature slacks for holding hot jobs in next scheduling cycle. And the spatial variation crossing multiple cores provides another dimension of potential to schedule the jobs: the hot jobs could be allocated on the current cool cores.

However, job scheduling algorithms introduce significant thermal cycling effect, which stresses the chip materials to rupture and is a major factor harming the chip lifetime. The unnecessary context switches in job scheduling cause the temperature fluctuates significantly over time, accelerating the thermal cycling effect. Thus, blindly scheduling the hot jobs on the cool cores overlooks the chip reliability factors, and smarter algorithms should minimize the thermal cycling effect.

Several task scheduling algorithms have been proposed, such as the Random, Round-Robin, and the Power-Balancing. Although Round-Robin and Power-Balancing effectively balance the power or temperature and reduce the peak temperature, they fail to take the performance of the jobs and the reliability of the chips together as the primary objective.

In this paper, we propose ThresHot, a task scheduling algorithm on CMP, to minimize the performance degradation from the DTM thermal emergencies (the peak temperature of the core overshoots the threshold) and to improve the reliability, with trivial overhead introduced. Both the temporal and spatial potential for scheduling on CMP are leveraged. By collecting the current temperature and aggressively predicting the future power and temperature, the scheduling decisions are made to achieve best possible performance and to improve reliability. Moreover, temperature behavior is studied in the granularity of function units, and multiple function units those may generate peak temperature are handled properly in ThresHot.

The rest the paper is organized as follows: section 2 briefly review the related work of task scheduling in thermal management; section 3 discuss the motivation and implementation of ThresHot; section 4 explains how to set up the simulation environment and validate our algorithm; section 5 analyze the result; section 6 serves as the conclusion.

II. Related Work

There have been a number of proposals on OS-assisted thermal management for single core chips. The HybDTM [2] technique controls temperature by limiting the execution of a hot job once it enters an alarm zone. This is achieved by lowering the priority of the hot job so that the OS allocates fewer time-slices to it and gives cool jobs relatively more time-slices to execute. An ideal simulation study was performed in [3] to show the benefits of interleaving hot and cool job executions. Our work, on the contrary, targets at CMPs.

In the multicore domain, Choi et al. [4] discussed three different task schedulers, heat-balancing, deferred execution, and threading with cool-loops, to leverage temporal and spatial heat slacks among application threads. Performance is traded for better thermal behavior. Donald and Martonosi [1] looked at thread migration policy for chip multiprocessor designs. Their migration policy is a simple balancing scheme which we will show could increase the temperature variation on the die. Chong et al. [5] proposed a 3D MPSoC thermal optimization algorithm that conducts task assignment, scheduling, and voltage scaling for a set of real-time workloads.

As an consequence of the scheduling, thermal cycling may hurt the reliability. Rosing and etc al. [6] considered three failure mechanisms most commonly used in industry to predict MTTF (Mean time to failure): EM (electromigration, TDDB (time-dependent dielectric breakdown), and TC (thermal cycling). They built their system-level reliability model based on the three mechanisms. Furthermore, they developed a power management policy which meets system reliability constraints. Similarly in this paper, we take these reliability concerns.

To achieve multiple optimal objectives, A. Coskun et al. [7] utilized the ILP (integer linear programming) method to find the optimal task scheduling both in minimizing the thermal hotspots and large temperature gradients, achieved by counting both objectives in the ILP objective. However, the ILP method is limited by two factors: first, the information such as the dependence and execution time of the scheduled tasks graph should be available *a priori*; second, the constraints of the scheduling is just to meet the deadline and dependence of the graph. With large computation overhead, the ILP-based scheduling is static, different from our on-line approach to schedule general applications.

In [8], J. Yang et al. discussed the similar scheduling approach to leverage the natural discrepancies in the thermal behavior among different workloads. However, this work targets the single core architecture and only one function unit temperature is observed. However, on multi-core CMP chips, due to the complex inter- and intra-core interaction, even in a single core multiple function units may be the possible hotspots. In this paper, we observe and handle multiple function units in making scheduling decision.

This paper proposes the ThresHot scheduling algorithm that reduces thermal emergencies, temperature variations across the die, and improves the reliability, while keeping the performance high. Existing approaches can only achieve part of those objectives but not all. We also compared with other various possible schedulers and show that our proposed scheme outperforms them in all those aspects.

III. Task Scheduling in Thermal Management

A. Scheduling problem formulation

Thermal-aware job schedule algorithms reduce thermal emergencies and improve job throughput by exploiting the temperature slacks on cool cores to handle the hot jobs. In CMP, both temporal and spatial temperature slacks can be exploited, comparing to the only temporal temperature slack in monolithic processors. The *temperature slack* is the gap of the current core temperature to the preset threshold. The larger the slack is, the more possibilities that the hot jobs can be assigned on the cool cores to reduce thermal emergencies and improve job throughput. The temporal temperature slack variations are caused by the fluctuating power of the jobs on monolithic processors or CMP cores. The spatial temperature slack variations are caused by the imbalance of temperatures of CMP cores running different jobs/threads. Hence, the thermal-aware job schedule algorithm on CMP should exploit both the spatial and temporal temperature slack potential.

Besides performance improvement, the thermal-aware job schedule algorithms also need to mitigate the thermal cycling effect. The thermal cycling effect is determined by the occurrences of large ΔT (temperature swing). Scheduling jobs on CMP cores produces extra ΔT due to switching the hot and cool jobs on cores. Consider an extreme case, frequently swapping the hottest and coolest jobs on the cores causes fast and significant ΔT, resulting accelerated thermal cycling effect. Observations from several existing scheduling algorithms such as Round-Robin and Balancing show that not all job switches are necessary: even without some schedule actions, there are no DTM actions to penalize the performance. Thus, the smart scheduling algorithm should distinguish the unnecessary schedule actions to mitigate the thermal cycling effect.

Considering both the performance and reliability, the challenges to design the thermal-aware job schedule algorithms are: 1) the inaccurate schedule decisions may harm the performance and/or reliability (inaccuracies comes from estimating the future power and temperature); 2) trade-off of the two goals of improving performance and reliability, since performance improvement is maximized by scheduling jobs whenever there is benefit; 3) the computation overhead of the schedule algorithm should be minimized since it adds to the performance overhead. Another challenge of CMP schedule algorithms is the uncertainty of hottest function units of CMP cores. In CMP, more than one function units may generate the peak temperature of the core over different execution phases. Shown in Fig 1 of CMP floorplan based on the Pentium 4 Northwood processor, Integer Register File, Memory Controller, DTLB (Data Translation Lookaside Buffer), etc. are candidates of the hottest function units. Although there are multiple thermal sensors on chip, most traditional schedule algorithms only monitor the only possible hottest function unit and treat the

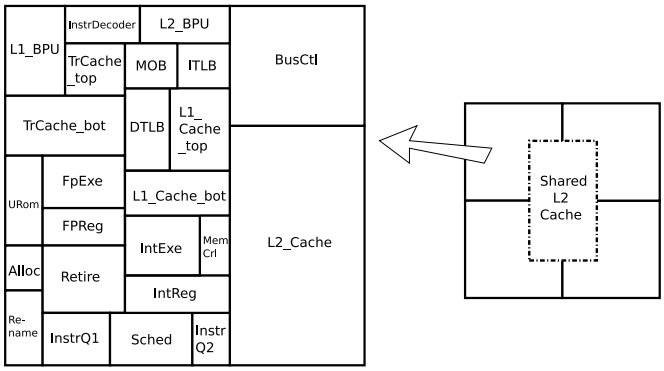


Fig. 1. Floor-plan of the Quad-Core: Multiple Function Units as Hotspots

core as a single node. Hence, the schedule algorithm for CMP should monitor multiple thermal sensors to generate correct schedule decisions.

Thus, the thermal-aware job schedule problem in CMP is formalized as: making consecutive decisions of assigning N heterogeneous jobs/threads on N homogeneous cores at scheduling points to achieve the three-fold objective:

- Minimize DTM actions triggered by thermal emergencies;
- Minimize the performance degradation of jobs;
- Minimize the thermal cycling effect.

B. Existing Scheduling Algorithms

Scheduling heterogeneous jobs on CMP has been a hot topic for years, and the proposed schedule algorithms include:

- Random: the schedule decisions are randomly assigning the jobs to the cores at each scheduling point.
- Round-Robin: the schedule decisions are generated such that all jobs are periodically and sequentially assigned to all the cores.
- Balancing: scheduling the jobs with ascending power density to the cores with descending temperature. Hence the cooler jobs are always running on the hotter cores and vice versa.

The Baseline scenario of the scheduling problem is that no schedule is applied to the CMP cores: all the jobs run concurrently and independently on fixed cores. In this paper, the Baseline scenario serves as the reference for comparison.

C. ThresHot Scheduling Algorithm

1) Motivation: Although all the above algorithms exploit certain potential to reduce DTM actions or thermal cycling, they fail to take both the performance and reliability as the primary objectives and thus can not achieve the three-fold objective. Moreover, the thermal information of the temperature and power of the function units of the cores is not utilized to generate smart schedule decisions. Therefore, we propose the new scheduling algorithm, ThresHot, to improve the performance of the jobs and reliability of the cores.

First, at each scheduling point, we try to verify the existence of the optimal decisions subset. For all the decisions in the optimal subset, there are no DTM actions and hence performance degradation for all the jobs in the subset during the current scheduling cycle. However, the optimal subset may not exist if several hot jobs are so hot that they could not be assigned to any core without triggering DTM. We define these hot jobs to be hot-hazard jobs, indicating that they are too hot and will definitely cause a thermal emergency. Then we classify the jobs into two groups: hot-hazard jobs and mild jobs. The hot-hazard jobs should be handled with higher priority in scheduling, since they introduce thermal emergencies and performance degradation. Although several mild jobs may trigger DTMs , the optimal decisions subset exists for the mild jobs.

To reduce the performance degradation caused by the hot-hazard jobs in the current scheduling cycle, we assign the hot-hazard jobs on coolest available cores. Although the hot-hazard jobs triggers the DTM action, the performance degradation is minimized. To show this effect, we split the scheduling cycle into two phases: 1) the transient phase from the start of the scheduling cycle to the point when the first DTM is triggered; 2) the penalizing phase in which DTM action is applied and the core runs in lower frequency. By assigning the hot-hazard jobs on coolest available cores, the temperature of the cores with the hot-hazard jobs experience a longer transient phase and result a shorter DTM penalizing phase in current scheduling cycle. Hence, the hot-hazard job performance in current scheduling cycle is improved.

Then, after scheduling the hot-hazard jobs, ThresHot generates the optimal subset for the remaining mild jobs. In the current scheduling cycle, the performance of the mild jobs is not penalized since there is no DTM action. Then, we only need to make schedule decisions to improve the performance of the job in next scheduling cycle and to reduce the thermal cycling effect.

We found that the two above goals can be achieved together by creating as large spatial temperature slacks as possible at the end of current scheduling cycle for the mild jobs. To enlarge the spatial temperature slack, we assign remaining hot jobs in the optimal decisions subset on hot cores to boost the temperature of the hot core as much as possible and still under the threshold temperature. Consequently, the cool jobs are left to be assigned to the cool cores. In this way, running cool jobs on cool cores yield coolest possible core at the end of the current scheduling cycle. Then, the hot-hazard jobs in next scheduling cycle will benefit from the reserved cooler cores, which improves the performance in next scheduling cycle. For the thermal cycling, creating large temperature slack inherently maintains the mapping of the jobs on the cores since hot jobs have already warm up the cores to be hot. Thus, the unnecessary switches are decreased to minimize the thermal cycling effect.

In estimating the peak temperature of the core at next scheduling point, we need to handle multiple hotspot function units. From our observation there are at most 4 function units to be the candidates of the hotspots with peak temperature in execution. For each core, we pick first 4 hottest function units

at current scheduling point, and then estimate the temperature of them at the next scheduling point. Then we pick the maximum of the peak temperature of the 4 function units to be the core peak temperature. In simulation, we verified that considering the first 4 hottest function units is accurate and sufficient to get the peak core temperature at next scheduling point.

2) ThresHot Algorithm: The ThresHot algorithm uses an aggressive approach to predict the power of current scheduling cycle, to schedule the jobs for current cycle such that performance both in current and next scheduling cycles are maximized, and to minimize the thermal cycling effect to improve the reliability, which achieves the 3-fold objective.

We design and build a special data structure, TSM (Temperature Slack Matrix), to implement the ThresHot algorithm. With TSM, it is simple and efficient to detect and schedule the hot-hazard jobs, and to schedule the jobs in the optimal subset to enlarge the spatial temperature slack. An example of TSM is shown in the Fig 2-(a).

To solve the problem of how to assign N jobs to N cores at scheduling point t, we build a $N \times N$ TSM. At scheduling point n, the element $S(i,j)$ in TSM is the temperature slack of core i at scheduling point $n+1$ with assigning the job j on core i in the scheduling cycle from n to $n+1$. Then the N^2 elements in the table stand for the temperature slack caused by N^2 possible decisions that the scheduler can make. Instead of searching the temperature of all the function units to get the peak core temperature, $S(i,j)$ is obtained by searching the minimum slack value from the picked 4 candidate function units. If $S(i,j) < 0$, the decision that assigning job j on core i causes the core i to overshoot the threshold and to trigger DTM action at least once in the current scheduling cycle. If $S(i,j) > 0$, the decision will not trigger the DTM action on core i. With smaller positive $S(i,j)$ value, the decision will generate higher temperature (tighter slack) and the temperature is under the threshold.

We use the TILTS [9] thermal model once per scheduling cycle to efficiently calculate the temperature of all the function units on the next scheduling point. Built on the HotSpot 4.0 [9] thermal model, the TILTS thermal model uses matrix multiplication to accelerate the temperature calculation. With the temperature at the current scheduling point and power of the current scheduling cycle, the temperature at the next scheduling point are calculated as the output of a thermal linear system. The power used for the current scheduling cycle is predicted as that of the last scheduling cycle. This simple power estimation method is verified to be within tolerance. Based on the TILTS thermal model, the temperature of all function units are calculated by combining two independent components in (1): future temperature determined by current temperature ($AT(n-1)$) and determined by the injected power in the current scheduling cycle ($BP(n-1)$):

$$T(n) = AT(n-1) + BP(n-1) \quad (1)$$

The vector $T(n-1), P(n-1)$ and $T(n)$ are the temperature at scheduling point $n-1$ (current temperature), power in scheduling cycle $n-1$ (current) and temperature at scheduling point n (future temperature) of all function units of CMP cores. A depicts the thermal resistance and capacitance in the heat dissipation. $AT(n-1)$ illustrates how the future temperature is determined by the natural heat dissipation based on the current temperature. And B depicts how the injected power $P(n-1)$ boost the temperature at the scheduling point n up by $BP(n-1)$. N^2 elements in TSM can be calculated efficiently by running TILTS once. We compute $BP(n-1)$ of each job and combine them with $AT(n-1)$ to generate the temperature slacks of N^2 dependent decisions. This simplification is achieved by leveraging two properties of B matrix, which reflects the temperature change determined by the schedule decisions:

- $B_{i,j} \approx 0$, for all $i \neq j$: the job that is running on core i has little influence on the temperature of core j.
- $B_{i,i} = B_{j,j}$, for all $i \neq j$: for the same job, the boosted temperature of each function unit caused by injected power is same for all cores, although the $AT(n-1)$ is dependent on the current temperature of the cores.

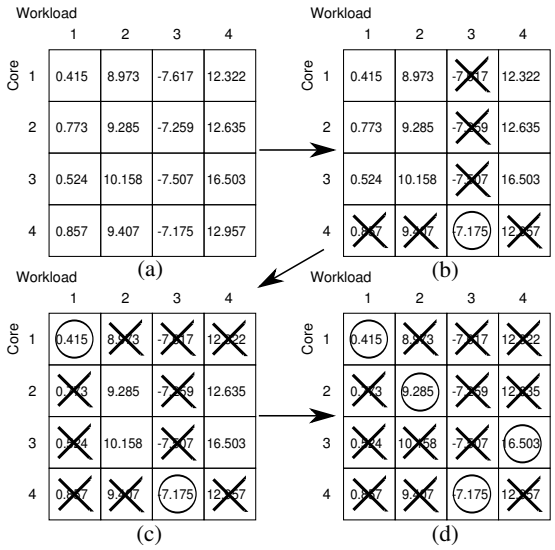


Fig. 2. ThresHot Scheduling Based on TSM

With the generated TSM, an example of making schedule decisions based on the TSM is illustrated in Fig 2. And the pseudo code is in Alg 1.

a All the decisions are valid.
b For hot-hazard $Job3$, assign $Job3$ on current coolest $Core4$, and invalid all the decisions associated with $Job3$ and $Core4$ (with cross). Then, all the remaining decisions does not trigger DTM, which comprises the optimal subset.
c Find the remaining decision with minimal positive $S(1,1) = 0.415$, assign $Job1$ on $Core1$, and invalid all the decisions associated with $Job1$ and $Core1$.
d Find the remaining decision with minimal positive $S(2,2) = 9.285$, assign $Job2$ on $Core2$, and invalid all the decisions associated with $Job2$ and $Core2$. The remaining $Job4$ is assigned to remaining $Core3$.

Algorithm 1 ThresHot Scheduling Algorithm

Require: Access the power and temperature from hardware (Performance Counter and Temperature Sensor) at scheduling point n.
Predict the power in the current scheduling cycle.
Assume no DTM for all cores at current scheduling point.
In TILTS, calculate the temperature of scheduling point $n+1$.
Generate the $N \times N$ matrix TSM.
Calculate the sum of the temperature slack of each job on each core.
Sort the sum of temperature slack with ascending order.
Set all the cores and jobs unoccupied.
for The job j with least temperature slack (hottest job) to the job with the largest temperature slack (coolest job) **do**
  if The $S(i,j) < 0$, for all i **then**
    Assign job j to the current coolest core.
    Mark the job j and core i occupied.
  end if
end for
Sort the $S(i,j)$, for all i, j that is unoccupied with ascend order
for all $S(i,j)_{unused}$ from small to large **do**
  For current $S(i,j)$, assign job j to core i
  Mark the job j and core i used
end for

D. Comparison

In this subsection, 5 algorithms: Baseline, Random, Round-Robin, Balancing and ThresHot are compared in the measure of DTM actions reduction, performance degradation, and thermal cycling effect.

In Baseline, since there is no job scheduling, no temporal and spatial temperature slack is exploited. Hence, the performance of the hot jobs are penalized most. However, there is no context switch overhead for Baseline configuration, resulting an optimal thermal cycling effect.

In Random, both the hot and cool jobs are treated evenly and arbitrarily. However, it is not smart enough to avoid the scenarios in which the hot jobs are assigned to hot cores, resulting more performance degradation than assigned to cool cores. Also, the context switch overhead and its influence on thermal cycling effect is random, dependent on the scheduling decisions.

In Round-Robin, intuitively, the power of each job is temporally evenly distributed on the N cores, resulting even spatial temperature. However, the temporal variation of the power of jobs cause that Round-Robin fails to achieve even power distribution at every scheduling point. Although DTM actions and performance degradation are reduced, the context switch overhead and the effect of thermal cycling for the Round-Robin is huge.

In Balancing, the decision is to balance the dynamic power on the cores, based on the collected thermal information of core temperature and the power of the jobs. The context switch overhead and the thermal cycling for balancing may be huge, depending on the variation of the temperature.

Overall, all the above existing algorithms except ThresHot just blindly schedule or intuitively evenly distribute the power of jobs to the cores for the current scheduling cycle, the performance and the thermal cycling effect are not handled. Moreover, the information of the power and temperature of the last scheduling cycle is scarcely utilized. Thus, they could not fully exploit the temporal and spatial temperature slacks in current and future scheduling cycles to improve performance and reliability. Although ThresHot algorithm is a bit more complex to implement, it yields best performance with minimal thermal cycling.

IV. Experiment Methodology

A. Experiment Setup

We use the HotSpot 4.0 [9] thermal model with the quad-core floorplan, and use TILTS to speedup the thermal simulation and to calculate the schedule decisions. All the simulations start with the steady state temperature after warming up.

To construct the quad-core floor-plan, we duplicate 4 Pentium 4 Northwood processors, and scale them to quarter area, as in Fig 1. The shared L2 cache locates at the center of the chip. Also, the power traces are scaled to $\frac{1}{3}$, which are originally collected from running benchmark on real Northwood processor with 8ms intervals. The power traces for each function unit are then calibrated. The power is perceived to be constant in the 8ms interval. In the quad-core thermal model, there are 93 function units for quad-core, with the L2 cache shared for 4 cores.

Then, we pack four benchmarks from the pool of 19 SPEC 2K benchmarks as one test suit. The benchmarks are classified by the type (Integer and Float Point), hottest function unit, peak temperature and peak temperature variation. Thus, we classify the test cases into several sub-cases: HHCC(2), HHMC(2), HMMC(2), HMCC(2), IIII(1), FFFF(1)[1].

In our scheme, the DTM is configured to be triggered on at the threshold temperature of $86.5\,^{\circ}\mathrm{C}$ and off at $85.5\,^{\circ}\mathrm{C}$. The frequency scaling coefficient is 0.7, and the voltage scaling coefficient is 0.92, resulting the power scaling coefficient to be 0.6286. The stall incurred by entering and exiting the DTM is $30\mu s$. The DTM scheme follows the independent distributed policy for each core. Although the scheduling cycle is 8ms, the time step for detecting and applying the DTM actions is set to be $80\mu s$, providing more accuracy.

B. Evaluation Metric

1) ETPP: Execution Time in Penalized Period: The DTM actions penalize the performance from DTM overhead (triggering-on/off and frequency scaling) and context switch overhead. In our experiment, four benchmarks in a test pack contains 2000 scheduling points with 8ms interval. With the

[1]H: Hot workload, M: Mild workload, C: Cool workload, I: Integer Benchmark, F: Floating Point Benchmark

penalty from the DTM actions, four benchmarks end individual execution at $(T_{alg1}, T_{alg2}, T_{alg3}, T_{alg4})$ (for Baseline, T_{basei}), and the TPP (Time in Penalized Period) of the alg is normalized to that of Baseline by:

$$ETPP_{alg} = \frac{\sum_{i=1}^{N} T_{algi} - 2000N}{\sum_{i=1}^{N} T_{basei} - 2000N}, N = 4 \qquad (2)$$

The performance overhead incurred by the thermal-aware job schedule algorithms include:

- DTM Triggering Overhead (10μs to 30μs).
- DTM Frequency Scaling Overhead.
- Task Migration Overhead.
- Task Scheduling Computation Overhead.

2) Reliability Improvement: The reliability of the chip is measured in MTTF (mean time to failure), which is affected by thermal cycling. The power management and the task scheduling will introduce frequent and significant thermal cycling. The number of thermal cycling to failure is determined by the occurrences and ΔT (the amplitude of the temperature swing) in the thermal cycling. Significant variations of temperature over time degrade the reliability a lot. Hence, we collected the occurrences ratio of ΔT over the execution for each job schedule algorithm.

V. Results

A. Results for Performance and Reliability

As in Fig 3, the $ETPP$ of 5 task scheduling algorithms: Balancing, Random, RoundRobin1, RoundRobin2[2], and ThresHot are shown. On average, the ThresHot saves 13% execution time in penalized period, while the Balancing saves around 8% and RoundRobin saves around 3%. Then, the ETPP is further split into two stacks, the DTM action overhead and switching overhead. The ThresHot algorithm effectively reduce the DTM actions and frequency scaling overhead, especially in the benchmarks with significant spatial temperature variation, since ThresHot fully exploits the spatial temperature slacks. In some benchmarks the DTM action overhead of ThresHot is larger than other algorithms. The reason is the imprecise power prediction and consequent wrong schedule desitions. For the benchmarks with significant temporal power variations, more errors are introduced in the calculating the TSM. Overall, even with the imperfections, the average DTM overhead of the ThresHot is the smallest among all the algorithms.

Also, the switching overhead of ThresHot is also the smallest of all the algorithms. Balancing and two Round-Robin algorithms always try to frequently and blindly migrate the jobs on the cores. On the other hand, ThresHot tries to enlarge the spatial temperature gap as long as the performance is not hurt, and the cool job could remain on the cool core as long as the hot job does not trigger DTM, resulting reduced context switches effectively.

Besides reducing the context switching overhead, the minimal context switching also reduces the thermal cycling and improves reliability. In Table I, the distribution of ΔT occurrences is shown. ThresHot is least likely to generate significant ΔT among all the scheduling algorithms. Without frequently swapping jobs on cores, the temporal variation of the cores due to the task scheduling is reduced.

[2]RoundRobin1 and RoundRobin2 is different from the task queue

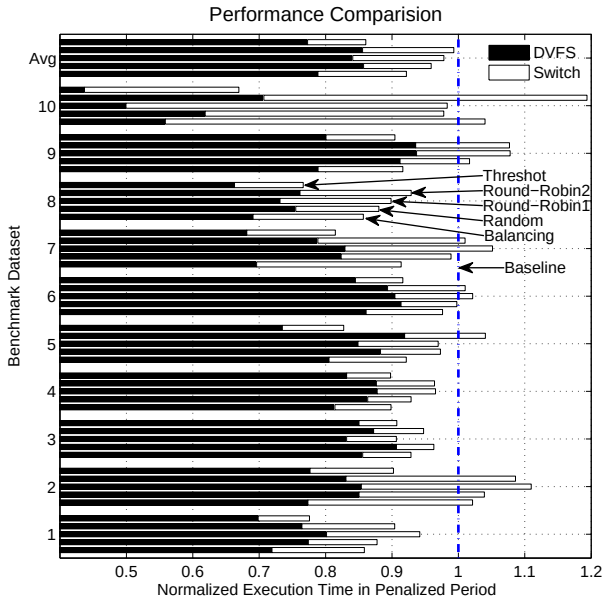


Fig. 3. Performance Comparison of Different Algorithms

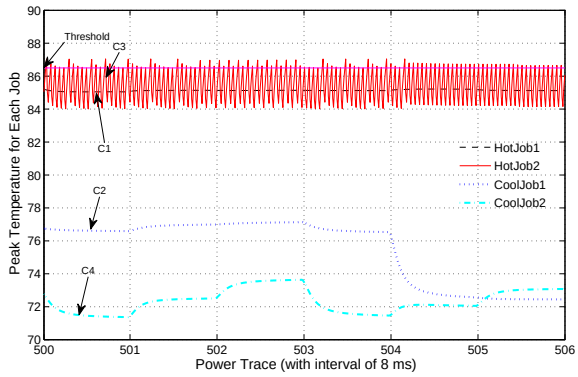
B. Thermal Behavior Analysis

The excerpt of the thermal behavior of the 6 scheduling algorithms: Base, Random, RoundRobin1, RoundRobin2, Balancing and ThresHot on the temperature behavior are shown in the Fig 4. In each sub-figure, 6 consecutive scheduling cycles (8ms each) are shown. For clarity, there are 100 finer intervals in each 8ms (the granularity of the DTM detecting intervals) scheduling cycle.

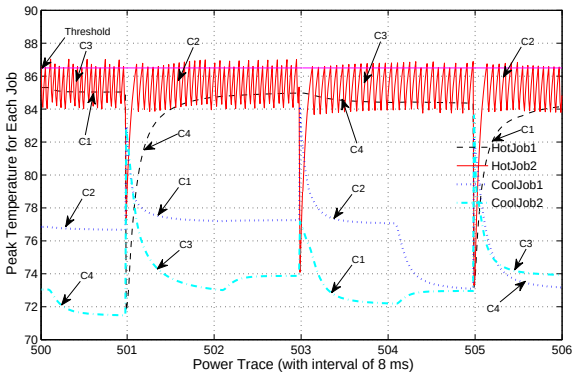
In Fig 4, all the temperature traces are classified into two distinct phases: transient phase without DTM actions from the scheduling point, and the oscillating phase from first DTM action. In the former phase, the temperature rises from the temperature at the scheduling point and approaches to the steady temperature determined by the power in the current scheduling cycle. If the steady temperature of the job is higher than the threshold, the oscillating phase is entered at some

TABLE I
Distribution(in percentage) of temporal variations

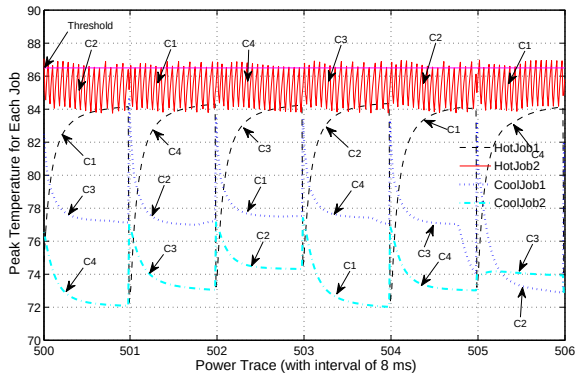
Algorithm	<10°C	[10°C, 15°C]	[15°C, 20°C]	>20°C
Baseline	99.91	0.07	0.02	0.01
Random	97.45	1.55	0.68	0.32
Balancing	95.50	2.67	1.23	0.60
Round-Robin1	95.83	2.60	1.05	0.52
Round-Robin2	96.91	1.93	0.78	0.38
ThresHot	98.22	1.21	0.43	0.14



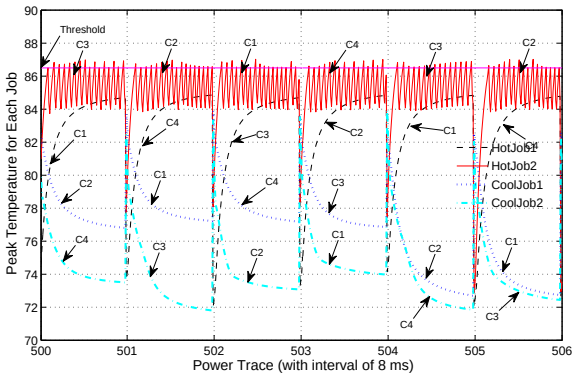
(a) Base



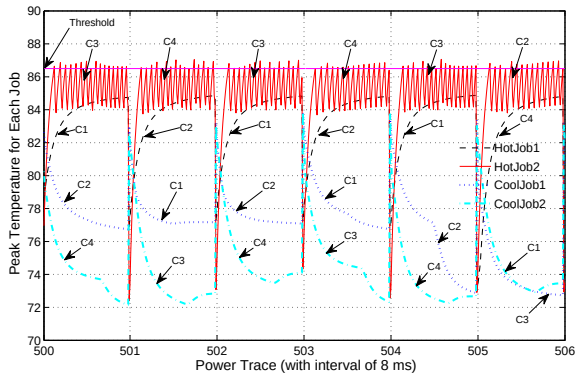
(b) Random



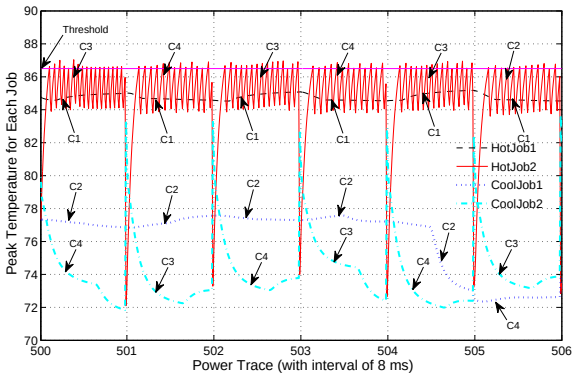
(c) RoundRobin1



(d) RoundRobin2



(e) Balancing



(f) ThresHot

Fig. 4. Thermal behavior of 6 scheduling algorithms

point in the current scheduling cycle. In the latter phase, the temperature experiences high-frequency fluctuation between the turning-off and turning-on threshold temperature. All the temperatures are referred by each job, and the core on which the job is running is marked by *Cx*.

We collect the power from real processors with 8ms intervals. In the simulation, the DTM actions slow down the execution progress of the job, resulting the delay of certain portion of the power in the current scheduling cycle to the next scheduling cycle. We model this effect in our simulation by linearly combining the remaining power of last cycle and the power of current cycle. Then, we will discuss how the scheduling algorithms impact the temperature of the CMP and the context switch.

As shown in Fig 4(a), there is no job switching in the Baseline design. Thus, the temperature of each job reflects its intrinsic properties determined by the power, such as the 3 jobs ($HotJob1$, $CoolJob1\&2$) without oscillating phase. However, the $HotJob2$ can not utilize the possible temperature slacks of other cool cores. Thus, the hot jobs under the oscillating phase suffer performance degradation in all execution period. There is no context switch overhead, and the thermal cycling

effect is minimal.

In Fig 4(b), the Random algorithm randomly assigns the jobs to the cores on every scheduling point. The Random takes advantage when the hot jobs are occasionally assigned to cool cores, such as the case in 501 point. Although the randomness can remove some DTM actions when the hot jobs are scattered across the cores, but it could not handle all similar cases in the consistent manner. Even worse, it may randomly continuously assign the hot jobs on the hot cores, resulting similar cases as in Baseline. Also, the context switch overhead is random.

In Fig 4(c) and 4(d), two Round-Robin algorithms assign the jobs on the cores in the Round-Robin way. The difference of the two algorithms is the job queue: the hot jobs are aggregated in the 4(c), and the hot jobs and cool jobs are interleaved in 4(d). In both cases, the cores will iterate the temperature caused by 4 jobs periodically. As in 4(c), there are 2 hot jobs: $HotJob1$ and $HotJob2$. The $HotJob2$ is assigned on the core on which $HotJob2$ was running on, thus the $HotJob1$ could not utilize the temperature slack of the cool cores. However, the $HotJob1$ can fully utilize temperature slack of the core on which the cool job just ran. Thus, performance degradation of the two hot jobs are maximized. On the other hand, as in 4(d), both the $HotJob1$ and $HotJob2$ can be assigned to cool cores, resulting a minimized difference of the performance degradation. However, in both cases, the hot job can not be always put on the cooler cores. As for the context switch overhead, the RoundRobin suffers most since all the jobs are rescheduled on each scheduling point.

In Fig 4(e), the Balancing algorithm tries to balance the power of each core to achieve an evenly distributed temperature. As we can see, the hot jobs are always assigned to the cool cores. Thus, the hot job can experience the longest possible transient phase, and the performance degradation of the hot job for the current cycle can be minimized. However, the jobs with mild temperature are also switched to balance the power, which introduces unnecessary context switches. Even worse, if the power difference of the jobs are not large enough, Balancing algorithm has little potential to further balance the temperature. As for the context switch overhead, the Balancing algorithm suffers almost as RoundRobin, since all the hot, mild and cool jobs are to be rescheduled on each scheduling point.

In Fig 4(f), the ThresHot algorithm schedule the jobs on the cores in a smarter way. It first tries to find whether all the jobs could be assigned on the cores without DTM actions. In 4(f), the $HotJob2$ triggers DTMs no matter what core it is assigned to. Thus, the $HotJob2$ is assigned on the current coolest core. For the remaining jobs those do not trigger DTMs, the schedule decisions are made such that the temperature gaps are enlarged as much as possible. Since the temperature gaps are the intrinsic results of the power gap of different jobs, the jobs are assigned such that the power gaps are maintained. In this way, the context switch is minimized, as long as no DTM actions occur, as shown in 501-505 in 4(f). Thus, ThresHot can both minimizing the DTMs of the hot jobs and context switches, resulting better performance and reliability.

C. Computation Overhead of ThresHot algorithm

The computation overhead of ThresHot is optimized by running TILTS once, and combining the $BP(n-1)$ portions of different jobs on $At(n-1)$ to generate the TSM. The main computation overhead is the matrix multiplication, and the computation in the decision phase is trivial. Assuming the current temperature can be obtained from the hardware sensor, and only the temperature of up to 4 function units for each core is required for the decision making phase, we measured the computation time to be $\approx 25.15\mu s$. This overhead includes both the matrix computation and decision phase. This computation overhead is not included in the later performance results, since we believe this computation can be performed efficiently on a dedicated assisting hardware block.

VI. Conclusion

We propose an aggressive thermal scheduling algorithm, with the object to reduce the thermal emergencies, to boost the performance and to increase the chip reliability. By fully exploiting the spatial and temporal temperature slacks and utilizing the thermal information of temperature and power, ThresHot responds well to varying temporal power and spatial temperature, and generates the scheduling decisions to improve both the current and next scheduling cycle. For the SPEC2K benchmarks, the ThresHot algorithm reduces 13% execution time in the penalized period, compared to 8% for Balanced and 3% for RoundRobin algorithm. Also, by removing unnecessary context switches, the thermal cycling effect are minimal among all the existing scheduling algorithms, which improves the chip reliability.

References

[1] D. Donald and M. Martonosi, "Techniques for multicore thermal management: classification and new exploration," in *ISCA*. The 33rd International Symposium on Computer Architecture, 2006, pp. 78–88.

[2] A. Kumar, L. Shang, L. S. Peh, and N. Jha, "Hybdtm: A coordinated hardware-software approach for dynamic thermal management," in *DAC*, 2006, pp. 548–553.

[3] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P. Bose, "Investigating the effects of task scheduling on thermal behavior," in *the 3rd Workshop on Temperature-Aware Computer Systems*. Held in conjunction with ISCA-33, 2006.

[4] J. Choi, "Thermal-aware task scheduling at the system software level," in *ISLPED'07*, Portland, August 27-29 2007, pp. 213–218.

[5] S. Chong, L. Shang, and R. P. Dick, "Three-demensional multi-processor system-on-chip thermal optimization," in *Proc. International Conference Hardware/Software Codesign and System Synthesis*, September 2007.

[6] T. S. Rosing, K. Mihic, and G. D. Micheli, "Power and reliability management of socs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 391–403, Aprial 2007.

[7] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Temperature-aware mpsoc scheduling for reducing hot spots and gradients," in *Proceedings of Asia and South Pacific Design Automation Conference*. University of California, San Diego, 2008.

[8] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *ISPASS*, 2008, pp. 191–201.

[9] Y. Han, I. Koren, and C. M. Krishna, "Tilts: A fast architecural-level transient thermal simulation method," in *Journal of Low Power Electronics*, 2007, pp. 13–21.