# Dynamic Thermal Management through Task Scheduling*

**Jun Yang**[†] **Xiuyi Zhou**[†] **Marek Chrobak**[¶] **Youtao Zhang**[§] **Lingling Jin**[‡]

[†]Electrical and Computer Engineering
[§]Computer Science
University of Pittsburgh, Pittsburgh PA 15261

[¶]Computer Science
University of California, Riverside
Riverside, CA 92521

[‡]Nvidia Corporate
Santa Clara, CA 95050

## Abstract

*The evolution of microprocessors has been hindered by their increasing power consumption and the heat generation speed on-die. High temperature impairs the processor's reliability and reduces its lifetime. While hardware level dynamic thermal management (DTM) techniques, such as voltage and frequency scaling, can effectively lower the chip temperature when it surpasses the thermal threshold, they inevitably come at the cost of performance degradation.*

*We propose an OS level technique that performs thermal-aware job scheduling to reduce the number of thermal trespasses. Our scheduler reduces the amount of hardware DTMs and achieves higher performance while keeping the temperature low. Our methods leverage the natural discrepancies in thermal behavior among different workloads, and schedule them to keep the chip temperature below a given budget. We develop a heuristic algorithm based on the observation that there is a difference in the resulting temperature when a hot and a cool job are executed in a different order. To evaluate our scheduling algorithms, we developed a lightweight runtime temperature monitor to enable informed scheduling decisions. We have implemented our scheduling algorithm and the entire temperature monitoring framework in the Linux kernel. Our proposed scheduler can remove 10.5-73.6% of the hardware DTMs in various combinations of workloads in a medium thermal environment. As a result, the CPU throughput was improved by up to 7.6% (4.1% on average) even under a severe thermal environment.*

## 1 Introduction

As technology for microprocessors enters the nanometer regime, power density has become one of the major constraints to attainable processor performance. High temperatures jeopardize the reliability of the chip and significantly impact its performance. The immense spatial and temporal variation of chip temperature also creates great challenges to cooling and packaging which, for the sake of cost-effectiveness [43], are designed for typical, not worst-case, thermal condition. This entails dynamic thermal managements (DTM) to regulate chip temperature at runtime.

There have been plenty of researches on DTMs at the microarchitecture level [6, 11, 16, 25, 28, 34, 35, 36]. Architecture solutions can respond to thermal crisis rapidly and reduce the chip temperature effectively through various performance reduction mechanisms.

Recently, a number of works have shown great potential in OS-assisted workload scheduling in addition to the hardware level techniques [7, 10, 14, 22, 23, 31]. The main approach is to leverage the temperature variations between different jobs, and swap them at an appropriate time to control the chip temperature. This has been practiced in both CMPs [7, 10, 31] and single-core processors [14, 22, 23]. Our work continues this direction of research.

We develop a heuristic scheduling algorithm to alleviate the thermal pressure of a processor. Our algorithm ThreshHot is based on the observation that, given two jobs, one hot and one cool, executing the hot job before the cool one results in a lower final temperature than after the reversed order. Thus, as long as executing the hot job itself does not violate the thermal threshold, the hot-cold order is better (or, at least, not worse) than the cold-hold order. Consequently, ThreshHot selects at each step the hottest job that does not exceed the thermal threshold.

ThreshHot outperforms other scheduling algorithms such as the one that changes the priority ranks of the hot and the cool jobs [22]. To know which job will be hot or cool for the hotspot, we develop a highly efficient *on-line* temperature estimator leveraging the performance counter based power estimation [19, 20, 22], compact thermal modeling [35], and a fast temperature solver [12]. We implemented the estimator for a Pentium 4 processor, although our general methodology is applicable to other processors such as CMPs. We calibrate and validate the model parameters against real measurements on our processor package. We also implemented our scheduling heuristics in the Linux kernel, together with our temperature estimator, and we tested the entire framework over the complete executions of SPEC CPU2K benchmarks, mediabench, packetbench and netbench. ThreshHot can remove up to 73.6% (34.5% on average) hardware DTMs in a medium thermal environment. With all the context switching, temperature estimation, and the thermal-aware scheduling overheads considered, ThreshHot consistently improves the performances of a mix of hot and cool programs by up to 7.2% (4.7% on average) compared to a base case with traditional thermal-oblivious Linux task scheduling. Our scheduling algorithm targets only batch jobs and thus has unnoticeable impact on interactive jobs and no impact on real-time applications.

The remainder of the paper is organized as follows. Section 2 discusses previous related works. Section 3 elaborates on our thermal-aware heuristic algorithm through mathematical derivations. Section 4 explains how to obtain online power and thermal information for our scheduler to work properly. Section 5 introduces our modifications of the Linux kernel scheduler. Section 6 compares our proposed scheduler with other alternatives. Section 7 reports the experimental results comparing ThreshHot to three other algorithms. Section 8 concludes this paper.

## 2 Prior Work

Some recent works have developed temperature control techniques for regular [32] and *real-time* [1, 2, 39, 40] workloads. The main approach is to dynamically adjust the CPU speed to minimize the peak temperature of the CPU, subject to the constraint that all jobs finish by their deadlines. Similar ap-

*This work is supported in part by NSF grants CCF-0734339, CNS-0720595, OISE-0340752 and CCF-0641177.

proaches can be used to minimize the energy consumption for real-time systems [30, 41]. Temperature control through job scheduling has also been utilized to enhance the reliability of a processor [26]. In contrast, our objective is to maximize the performance by scheduling the workloads to keep the temperature below a given threshold. Note that the threshold can be the manufacturer-defined temperature threshold[1] for the physical chip, or an OS-defined threshold for a system to stay within a thermal envelope. Hence, we always attempt to run workloads with full speed as long as the temperature stays below the given threshold.

Thermal management through workload scheduling has been studied in various scenarios. In CMPs, the "Heat-and-Run" technique performs thread assignment and migration to *balance* the chip temperature at runtime [31]. In another work [10], a suite of DTM techniques, job migration policies, and control granularity are jointly investigated to achieve the maximum chip throughput. Also recently, a simple periodic thread swapping between two cores to balance the chip temperature was studied on a dual-core processor [7]. All these approaches exploit simple interleaving between hot and cool jobs to improve thermal characteristics of a schedule. Our objective is to find the best thread for a core when it becomes hot, and this thread may not be the coolest available thread. For example, when there is both a medium hot and a cool thread, our scheduler will pick a medium hot thread as long as it will not trigger DTM. In this paper, we demonstrate this philosophy using a scheduling heuristic on a single-core processor, and leave its extensions to CMPs as future work.

In single-core domain, the "HybDTM" [22] controls temperature by limiting the execution of the hot job once it enters an alarm region. This is achieved by lowering the priority of the hot job so that the OS allocates it with fewer time slices to reduce the processor temperature. The same principle can be seen in [4], where energy dissipation rate is evened among hot and cool jobs through assigning different CPU time to them. Our technique does not modify the time allocated to hot and cool jobs, as this would affect the fairness policy of the system. Instead, we attempt to rearrange their execution order within each OS epoch to lower the overall temperature. This allows us to control the temperature while preserving priorities among different jobs.

Thermal control through workload management has also been studied at the system level. In [29], a temperature-aware workload placement heuristic was studied for data centers to minimize the cost of cooling. The "Mercury and Freon" [15] framework uses a software to estimate temperatures for a server cluster and manages its component temperatures through a thermal-aware load balancer. The "ThermoStat" [8] tool employs a detailed 3D computational fluid dynamics model for a rack-mounted server system. This tool can guide the design of better dynamic thermal management techniques for server racks. Our work targets at CPU temperature control, which can be complementary to system level thermal management schemes.

# 3 Thermal Scheduling Algorithms

When the processor overheated and forced to slow down, nearly all vital measures will be degraded: throughput and utilization will be reduced, response time will increase, jobs are more likely to miss deadlines, etc. Thus, independently of the characteristics and focus of a given system, processor overheating will negatively affect its performance.

When incorporating new features, such as thermal awareness, into a scheduler, it is desirable to make them as transparent to the user as possible; in particular, to keep the existing scheduler structure and properties. For this reason, we focus our work on a batch system for which the main objectives are the minimum turnaround time, maximum throughput, and CPU utilization. For batch jobs, the OS periodically interrupts the job execution to maintain its statistics and determines if a different job should be swapped in and, if so, which one. We amend the decision of which job should be selected next with thermal-awareness while keeping all other features intact. Therefore, in every scheduling interval, the OS needs to select the best job anticipating that such a selection would lead to the overall least amount of thermal violations.

## 3.1 The Principle

To keep the temperature below the threshold, the naïve, greedy approach would be to minimize the *current* chip temperature by executing at each step the coolest available job. As a result, the jobs are scheduled in the order of increasing temperature, from coolest to hottest. As it turns out, however, the greedy schedule actually increases the chances of exceeding the temperature threshold in the long run. To see this, consider a simple case where at some schedule interval $t$ only two jobs $x$ and $y$ are available, with power consumption $P_x$ and $P_y$ respectively, where $P_x < P_y$ (so $x$ is cooler than $y$). We will show that if we execute these jobs in order $xy$ ($x$ before $y$, as in the greedy schedule) then the temperature at the end of $t+1$ is higher than for the order $yx$ ($y$ before $x$). This means that **if the temperatures for both orders stay below the threshold**, then order $yx$ is less likely to cause a DTM in the future.

Consider the simplified thermal model for a processor treated as a single node. The duality between heat transfer and electrical phenomena [21] has provided a convenient basis for modeling the chip temperature using a *dynamic compact thermal model* [35]:

$$\frac{1}{R}T + C\frac{dT}{dt} = P, \tag{1}$$

where $T$ is the temperature relative to the ambient air, $R$ and $C$ are, respectively, the chip's effective vertical thermal resistor and capacitor, and $P$ is the power consumption. Note that when $\frac{dT}{dt} = 0$, the chip reaches its steady temperature $RP$ which depends on the average power of a job. The time to reach the steady temperature is determined by the RC constant ($R \times C$) of the thermal circuit. However, when the chip is switching among different jobs prior to the steady temperature, it is always in a transient stage (i.e. $\frac{dT}{dt} \neq 0$).

Discretizing the time scale into small time steps $\Delta t$ and denoting by $T_i$ the temperature at time $i\Delta t$, Equation (1) can be approximated by

$$\frac{1}{R}T_i + C\frac{T_i - T_{i-1}}{\Delta t} = P. \tag{2}$$

Rearranging the terms, we have $T_i = \alpha T_{i-1} + \beta P$, where $\alpha = \frac{RC}{\Delta t + RC}$ and $\beta = \frac{R\Delta t}{\Delta t + RC}$ are constants dependent on $\Delta t$ and, clearly, $\alpha < 1$. If each scheduling interval is divided into $n$ steps of length $\Delta t$, the temperature at the end of this interval can be expressed as:

$$T_n = \alpha^n T_0 + (\alpha^{n-1} + \alpha^{n-2} + \cdots + 1)\beta P. \tag{3}$$

[1]This threshold is a safe operating temperature beyond which the chip might be damaged due to overheating, and exceeding it triggers DTMs.

For schedule $xy$, the temperature after completing $y$ ($2n$ steps) will be

$$T_{2n}^{xy} = \alpha^{2n} T_0 + (\alpha^{n-1} + \alpha^{n-2} + \cdots + 1)\beta(\alpha^n P_x + P_y), \quad (4)$$

while for schedule $yx$, this final temperature will be

$$T_{2n}^{yx} = \alpha^{2n} T_0 + (\alpha^{n-1} + \alpha^{n-2} + \cdots + 1)\beta(\alpha^n P_y + P_x). \quad (5)$$

It is now easy to see that $P_x < P_y$ implies $T_{2n}^{yx} < T_{2n}^{xy}$. That is, *scheduling the hotter job first results in a lower final temperature*. We emphasize that this is benecitial **only when running the hotter job first does not increase the temperature above the threshold**. Figure 1 gives an intuitive illustration of the impact of scheduling on temperature. The graph shows temperature variation for the IntReg unit with two different power inputs, representing two different jobs. They are scheduled in two different orders as just described. The graph was obtained using a full-chip thermal model (rather than a single node as a whole) solved by the fourth order Runge-Kutta method. As we can see, running the hotter job first results in lower final temperature. If the chip's thermal threshold is in between the difference of the two ending temperatures, the greedy schedule would cause a thermal violation.

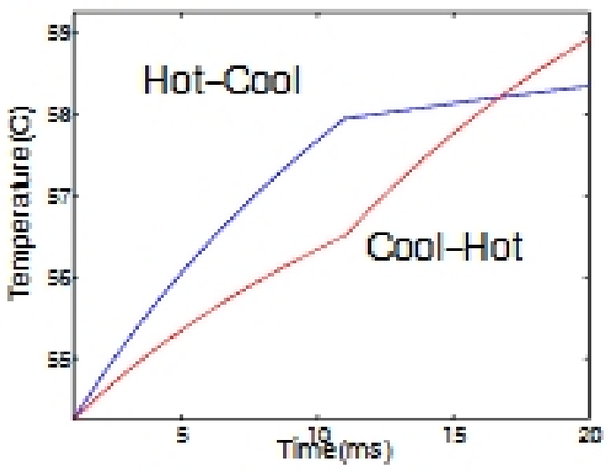


**Figure 1. The impact of scheduling a hot and cool program in different orders.**

Suppose now you are given a schedule for some number of job intervals. Suppose further that in this schedule there are two consecutive job intervals $x$, $y$ with $x$ before $y$, such that $P_x < P_y$ and that executing $y$ first will not exceed the threshold. Then, by the reasoning above, we can exchange $x$ with $y$, and this swap will not increase the number of thermal violations in the whole schedule. The reason is that in this new schedule, after completing $yx$, the temperature will be lower than in the original schedule after completing $xy$, so we cannot cause an increase of the temperature at any given step later in the schedule. This observation naturally leads to the following heuristic:

$\mathcal{P}$: *at each step choose the hottest job* ***that will not increase the temperature above the threshold***.

The above policy $\mathcal{P}$ is the basis of our algorithm ThreshHot. We emphasize that $\mathcal{P}$ does not guarantee to minimize the total number of thermal violations for a given set of job intervals (in fact, in a more rigorous setting, this problem can be shown to be NP-hard [9]); nevertheless, it computes a local minimum and it constitutes a reasonable heuristic for our application.

We also need to address the case when no job interval satisfies policy $\mathcal{P}$, i.e. all the jobs would increase the temperature above the threshold. In this case, it is most beneficial to pick the *hottest* job interval for execution. This is because the hardware thermal management (e.g. DVFS) will be triggered to cool the chip regardless of which job we choose, and selecting the hottest job interval at this time reduces the likelihood of a future thermal violation.

For example, suppose there are three job intervals available, say $J_1$, $J_2$ and $J_3$ with descending powers. If picking $J_1$ would increase the temperature above the threshold while picking $J_2$ would not, then policy $\mathcal{P}$ will first pick $J_2$ to run. If all of them would exceed the threshold, $\mathcal{P}$ will pick $J_1$.

We remark here that the OS fairness policy imposes some restrictions on how long a job interval can be postponed (this will be discussed in more depth in Section 5). Thus, in addition to the rules described above, the choice of the next job to run must be consistent with these fairness restrictions.

## 3.2 In Practice

In the earlier discussion we assumed a simple case where the CPU is considered as a single node and the heat is only dissipated through the vertical thermal resistor and capacitor. In reality, there is a great temperature variation on-die and only the temperature at the hottest spot should be maintained below the threshold. This scenario is more complex than for a single node, as the heat can also be dissipated laterally. Therefore, the thermal model in Equation (1) will be expanded into a matrix form in which every node is described by:

$$\frac{T - T_1}{R_{L1}} + \frac{T - T_2}{R_{L2}} + \frac{T - T_3}{R_{L3}} + \frac{T - T_4}{R_{L4}} + \frac{T}{R} + C\frac{dT}{dt} = P \quad (6)$$

where the first four extra terms describe the heat transfer from the central node (with temperature $T$) to its lateral neighbor nodes (with temperature $T_1$-$T_4$). The number of neighbors per node depends on the processor floorplan and how the system is discretized. We have shown four nodes as an example, with $T_i$ being the temperature for the $i^{th}$ neighbor node, and $R_{Li}$ being its lateral resistance from the central node.

The temperature $T$ of the hottest spot on-chip, described by Equation (6), is higher than the $T_i$'s. Also, heat is removed mostly from the vertical path and less from the surface [10, 31, 35]. In more quantitative terms, our experience with a full-chip model shows that the $R_{Li}$'s are typically 10∼20 times the $R$ for a hot unit such as the IntReg. The resulting lateral RC time constants are on the order of 100 milliseconds and vertical RC time constant is less than 10 milliseconds. Since the left hand side of Equation (6) is dominated by the last two terms, we can still treat a hotspot as a single node as before.

# 4 Obtaining Power and Temperature Online

As discussed above, our thermal-aware scheduling algorithm needs information about the peak temperature of the processor and power usage for the executed jobs. In this section, we explain how these values can be computed at runtime.

## 4.1 Computing the Temperatures

Most current processors are equipped with an on-chip thermal sensor for detecting the chip temperature at runtime. The sensor compares the current temperature with a preset threshold and signals a violation if the former is higher. The hardware then responds to such a signal immediately by throttling the performance so that the chip generates less power and, as a result, the temperature starts to drop. In Pentium 4, for example, the performance is throttled by dynamic frequency scaling — the