Thermal-Aware Task Scheduling for 3D Multicore Processors

Xiuyi Zhou, Jun Yang, *Member*, *IEEE*, Yi Xu, *Student Member*, *IEEE*,
Youtao Zhang, *Member*, *IEEE*, and Jianhua Zhao, *Member*, *IEEE*

Abstract—A rising horizon in chip fabrication is the 3D integration technology. It stacks two or more dies vertically with a dense, high-speed interface to increase the device density and reduce the delay of interconnects significantly across the dies. However, a major challenge in 3D technology is the increased power density, which gives rise to the concern of heat dissipation within the processor. High temperatures trigger voltage and frequency throttlings in hardware, which degrade the chip performance. Moreover, high temperatures impair the processor's reliability and reduce its lifetime. To alleviate this problem, we propose in this paper an OS-level scheduling algorithm that performs thermal-aware task scheduling on a 3D chip. Our algorithm leverages the inherent thermal variations within and across different tasks, and schedules them to keep the chip temperature low. We observed that vertically adjacent dies have strong thermal correlations and the scheduler should consider them jointly. Compared with other intuitive algorithms such as a Random and a Round-Robin algorithm, our proposed algorithm brings lower peak temperature and average temperature on-chip. Moreover, it can remove, on average, 46 percent of thermal emergency time and result in 5.11 percent (4.78 percent) performance improvement over the base case on thermally homogeneous (heterogeneous) floorplans.

Index Terms—3D processors, thermal-aware scheduling.

1 Introduction

The 3D integration technology has gained significant attention recently. This is a technology that reduces wiring both within and across disparate dies, as wiring has become a major latency, area, and power overhead in modern microprocessors. Studies have shown that wires can consume more than 30 percent of the power within a 2D CMP [3]. 3D technology provides vertical stacking of two or more dies with a dense, high-speed interface, reducing the wire length by a factor of the square root of the number of layers used [16]. This significant reduction leads to improved performance and lower power dissipation on the interconnection.

One key challenge in 3D die stacking is the heat generation from the internal active layers because the power density per unit volume increases drastically in 3D. This exacerbates existing hotspots and can create new hotspots within the chip, especially when active logic circuits are vertically aligned. For example, the peak temperature can increase by 17-20°C in a two-layer 3D implementation for an Alpha-like processor, compared to a 2D design [14], [22]. Other studies on logic-logic stacking 3D floorplans [1], [3], [23] also show similar thermal constraint.

There exist dynamic thermal management (DTM) techniques such as dynamic voltage and frequency scaling (DVFS) at the architecture level to mitigate this problem. Hardware DTMs can respond to thermal crisis quickly and control the temperature efficiently by reducing the processor power, but inevitably lead to degraded performance. Recently, there has been an increasing interest in OS-assisted task scheduling on both single-core and 2D chip multiprocessors to alleviate the thermal condition on-chip [5], [8], [17], [18], [21]. OS-assisted task scheduling can reduce the number of times DTMs are triggered while still meeting the thermal constraint. This technique not only improves the chip performance under the same thermal constraint but also does not require any hardware modifications. Hardware DTMs are engaged only when task scheduling cannot keep the temperature below the thermal threshold.

In this paper, we propose a heuristic OS-level technique that performs thermal-aware task scheduling on a 3D chip multiprocessor (CMP). The proposed technique aims to improve task performance by keeping the temperature below the threshold to reduce the amount of DTMs. Unlike previous thermal-aware OS task scheduler for single-core or 2D CMP, our scheduler for 3D chips must take into account the thermal conduction in the vertical direction. Early studies have shown that vertically adjacent dies have strong thermal correlations [2], [26]. For example, a core in one layer could become hot because of a high-power task running in the same vertical column but at a different layer. Based on these observations, our proposed scheduler always considers the aggregated power of cores that is vertically aligned. Further, when a core is overheated, we choose to engage DTM on a vertically aligned core that generates the most power. Such an approach can greatly reduce the total power in one vertical column and quickly

- *X. Zhou is with the Department of Electrical and Computer Engineering, University of Pittsburgh, 348 Benedum Hall, Pittsburgh, PA 15261. E-mail: xiz44@pitt.edu.*
- *J. Yang is with the Department of Electrical and Computer Engineering, University of Pittsburgh, 336 Benedum, 3700 O'Hara Street, Pittsburgh, PA, 15261. E-mail: juy9@pitt.edu.*
- *Y. Xu is with the Department of Electrical and Computer Engineering, Swanson School of Engineering, University of Pittsburgh, 761A Benedum Hall, Pittsburgh, PA 15261. E-mail: yix13@pitt.edu.*
- *Y. Zhang is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: zhangyt@cs.pitt.edu.*
- *J. Zhao is with the Department of Computer Science and Technology and the State Key Laboratory of Novel Software Technology, Nanjing University, Hankou Road #22, Nanjing 210093, China. E-mail: zhaojh@nju.edu.cn.*

Manuscript received 15 Aug. 2008; revised 29 Jan. 2009; accepted 2 Feb. 2009; published online 11 Feb. 2009.
Recommended for acceptance by M. Ould-Khaoua.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-08-0309.
Digital Object Identifier no. 10.1109/TPDS.2009.27.

1045-9219/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

cool down the overheated core. Our experiments show that the proposed scheduler outperforms a Random and a Round-Robin scheduler. On average, we can remove 46 percent of hardware DTMs and obtain an improvement of 5.11 percent over the baseline on the thermally homogeneous floorplan. With an enhanced version featured with dynamic tuning scheme, we can remove 46 percent hardware DTMs which result in 4.78 percent performance improvement over the base case on the thermally heterogeneous floorplan.

The remainder of this paper is organized as follows: Section 2 discusses previous related works. Section 3 elaborates the motivation of our thermal-aware heuristic algorithms. Section 4 compares our proposed scheduling algorithm with other alternatives. Section 5 introduces the experimental methodology. Section 6 reports the results and compares different algorithms. Section 7 concludes the paper.

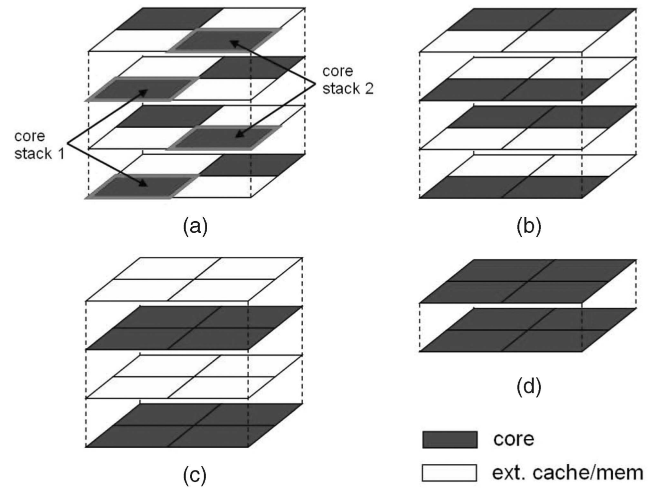


Fig. 1. 3D chip multiprocessor floorplan options.

2 Prior Work

There have been many works recently investigating the performance potential and the challenges in 3D CMP designs. Mysore et al. [20] proposed to stack on top of a normal processor a profiling die that can identify memory leakage, perform diagnosis etc., to save the area and power on the main die. Black et al. [3] studied the performance advantages and thermal challenges for stacking a large DRAM and SRAM cache on a processor as well as implementing a processor in two layers. Xie et al. [26] reported that the peak temperature in a 3D chip of two layers and one die per layer can be as high as 125°C. More importantly, there is only a difference of a couple of degrees, in the worst case, between the hotspots in the top die and the bottom die. This indicates a strong thermal correlation among adjacent layers in a 3D processor. To ensure better heat dissipation in a 3D chip, Puttaswamy and Loh proposed a "Thermal Herding" design [23] at microarchitecture level, which lowers the power of the chip by splitting individual function unit blocks across multiple layers and places the most frequently switched part, or activity, closest to the heat sink. Alternatively, adding thermal vias can also alleviate the thermal conditions within a 3D chip. Goplen and Sapatnekar [10] studied that the proper placement of thermal vias in 3D IC design can obtain a maximum of 47.1 percent reduction in temperature. In the multicore domain, Loh et al. [19] introduced different approaches for implementing single-core and multicore 3D processors. Particularly, they pointed out that stacking separate cores (in multicore design) can significantly reuse the existing 2D designs and the interface between the cores needs no more than a few thousand connections.

Compared to the previous work, this paper focuses mainly on software approaches to thermal management in 3D CMP. There have been proposals on OS-assisted thermal management for single-core chip. The HybDTM [17] 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 timeslices to it and gives cool jobs relatively more timeslices to execute. An ideal simulation study was performed in [18] to show the benefits of interleaving hot and cool job executions. However, neither performance study nor task switching overhead was considered. In the 2D multicore domain, the "Heat-and-Run" scheduler [21] assumes that there is always an idle and cool core present in a CMP such that an overheated core can migrate its thread to the cool core. However, in our technique, all cores are assumed to be busy and have temperatures above an idle core's temperature. Choi [5] compared and implemented three different task schedulers, heat balancing, deferred execution, and reducing threading with cool loops, to leverage temporal and spatial heat slacks among application threads. The proposed mechanisms are implemented in PowerPC 5. Chong et al. [6] proposed a 3D MPSoC thermal optimization algorithm that conducts task assignment, scheduling, and voltage scaling for a set of real-time workloads. The goal was to slow down the workloads as long as the deadlines are met. This is quite different from our approach, which focuses on best performance and low thermal profile.

3 Motivation and Rationale

3.1 A Representative Floorplan

There have been a number of 3D CMP floorplans, as shown in Figs. 1a, 1b, and 1c, proposed in the literature [1], [3], [19]. In these figures, cores are stacked on each other, with extended cache or memory in between. We observed that for a 3D stacked chip to be scalable in layer count, it is inevitable to encounter more than one active core in one vertical core column, no matter how the active cores and cache banks are placed in the floorplan. Further, if we look at the distance of each core stack to the heat sink (on either the top or bottom of the chip), we can classify these floorplans into two categories.

Figs. 1a and 1b represent the first category in which the distance of some core stacks, e.g., core stack 1 in (Fig. 1a), to the heat sink is different from others such as core stack 2. These floorplans are *thermally heterogeneous*, meaning that the heat dissipation property of different core stacks is different. For example, if the heat sink is on the bottom of the stacked chip (as illustrated in Fig. 2), core stack 2 is further away from the heat sink than core stack 1. Thus, heat dissipation for cores in stack 2 will be more difficult than those in stack 1. In contrast, Fig. 1c has a rather homogeneous thermal property because all cores are equally distant from the heat sink. Our preliminary work

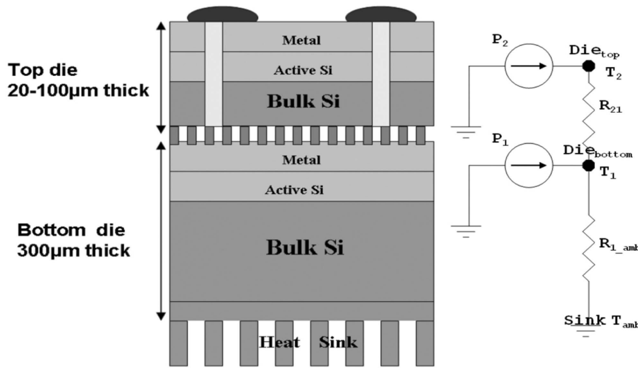


Fig. 2. A face-to-back 3D die stacking structure (adapted from [3]) and the corresponding thermal model.

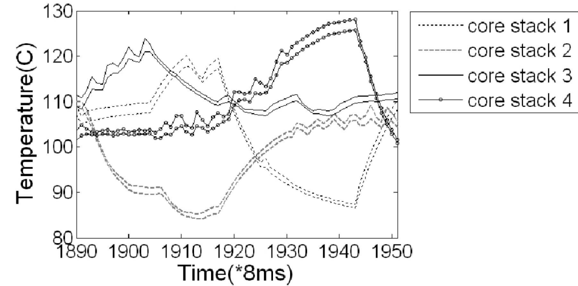


Fig. 3. Thermal correlation between the adjacent dies.

[28] focused only on homogeneous floorplans while this paper considers both.

Despite these distinctions among different floorplans, they still share some important property. The heat from any core can quickly propagate vertically to other cores above and below. For all these floorplans, the cache layers almost serve as heat conductance between the core layers. Considering this commonality among various 3D floorplans, we choose to use the floorplan in Fig. 1d as a representative to first introduce the general rationales behind our scheduling algorithms. Then, we will discuss details of our algorithms for homogeneous and heterogeneous floorplans, respectively. In Fig. 1d, there are two layers and each layer contains four cores. The cache banks are subsumed within each core.

3.2 Vertically Adjacent Layers Have Strong Thermal Correlations

Similar to a regular 2D processor where heat dissipates mostly in the vertical direction [13], 3D chips also have better heat conductivity in vertical than horizontal direction. This implies that vertically adjacent cores have larger thermal impact among each other than horizontally adjacent cores. We will use a simple heat transfer model to explain this phenomenon. Fig. 2 shows a basic two-layer 3D chip structure (adapted from Black et al. [3]). We use a face-to-back bonding technology for better scalability in layer count. The top layer is thinned for better electrical characteristics and improved physical construction of the through silicon vias for power delivery and I/O. A thin die also has better heat conductivity than a thick die such as the bottom die. As we can see, the distance between the two active silicon dies are very small (<100 μm). This directly determines the high heat conductivity between the two adjacent dies. The heat transfer model for the 3D chip is shown on the right of the figure. Here, one die is modeled using one node. Its temperature and power are denoted by T and P, respectively. R_{21} represents the thermal resistance between the two nodes. R_{1_amb} represents the thermal resistance between the bottom node and the ambient air. We omit the thermal capacitance here to model only the steady-state temperature. (In our experiments later, both thermal resistance and capacitance are modeled.) Let T_1 and T_2 be the temperatures (relative to the ambient air) in the bottom and top node, respectively. Then,

$$T_1 = R_{1_amb}(P_1 + P_2), \tag{1}$$

$$T_2 = R_{1_amb}(P_1 + P_2) + R_{21}P_2. \tag{2}$$

Hence, the temperature difference between the two nodes is $R_{21}P_2$. From the parameter used in the literature [3], [7], [13], [24], R_{21} is 0.0108-0.0159 K/W. P_2 represents the power generated by the entire die. This value is in the range of 40-70 W for a typical single-core processor. Therefore, the temperature difference between the top and bottom dies is merely 0.43-1.11 K.

Such a strong thermal correlation between the two adjacent dies can also be demonstrated from our simulation. Fig. 3 shows a typical thermal profile of running eight threads concurrently on eight cores as floorplaned in Fig. 1d (the experimental setup will be introduced in Section 5). Here, eight threads are eight different benchmarks chosen from the benchmark suite we use. We refer to vertically aligned two cores as a core stack. We can see in Fig. 3 that there are four distinct clusters of temperature curves. Each cluster has drastically different variations from others. However, each cluster has two lines that are very close to each other. Their variations are almost always synchronized. The four clusters correspond to the four core stacks in the floorplan. And the two lines in each cluster correspond to the temperature variation of the two cores per stack. This experiment shows clearly the strong correlation between the adjacent dies, as the temperatures for different core stacks hardly have any dependencies among them, but within each core stack, the temperatures of the two cores are strongly correlated. Such correlation can still be observed for a four-layer floorplan in our experiments, as the intermediate thin cache layers serve as good heat conductors among their vertical core neighbors.

3.3 The Die Layers Farther from the Heat Sink Are Usually Hotter

Not only are the cores in a stack strongly correlated in their temperatures but also the ones on the top are usually hotter than those near the bottom. This has also been noted in the literature for steady-state temperatures [2], [19]. For clarity, we refer to the cores farther from the heat sink as "top" cores, as illustrated in Fig. 2. The intuition is that the bottom cores are closer to the heat sink, therefore, their heat can be removed more quickly. Here, we give a more analytical analysis taking into account the thermal capacitance as well.

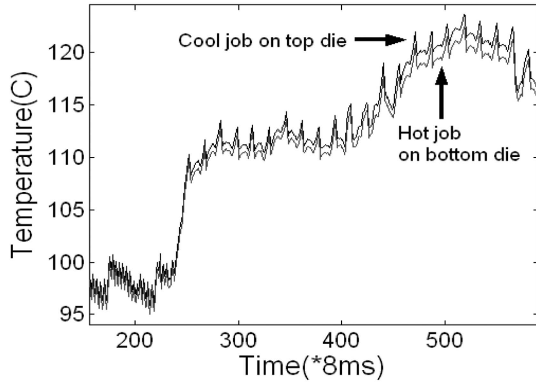


Fig. 4. Demonstration of the top die being hotter than the bottom die.

Suppose in the thermal model depicted in Fig. 2, the thermal capacitance between the top die and ambient air is C_2. Then,

$$\frac{T_2 - T_1}{R_{21}} = P_2 - C_2 \frac{dT_2}{dt}. \quad (3)$$

As mentioned earlier, P_2, which represents the power of a modern processor, has a typical value range of 40-70 W. C_2 represents how quickly temperature changes from the top die. For a thin die within 100 μm in a two-layer 3D chip, the thermal capacitance is reported as 23.6-37.4 $\mathrm{mW{\cdot}s/K}$ [3], [24]. dT_2/dt is the temperature change rate within a short time. From our experimental experience and many other results in the literature, temperature varies slowly with time. For example, we observed a less than 6°C increase in temperature in a 8 ms window using HotSpot 3.0.2 for 3D chips. Hence, the right-hand side of (3) is usually positive with a range of 12-52.3 W. Therefore, T_2 is usually higher than T_1.

We also performed simulations to testify the above observation. We intentionally put the coolest job (the job with the lowest average temperature in a 2D chip) in our benchmark suite on the top die and the hottest job on the bottom die in a 2-core-stacked 3D chip setting. The temperatures of the two cores are shown in Fig. 4. We can see that the top core has higher temperatures than the bottom layer almost always. Such an observation serves as a guideline to the development of our heuristic scheduling algorithm.

4 Scheduling Algorithms

The strong correlations among the cores in one stack lead to a scheduling that considers the entire stack as a whole. The fact that top cores are hotter than the bottom cores suggests that threads within a core stack should be placed with care. Furthermore, we take advantage of this observation and introduce a new voltage/frequency scaling mechanism that results in the fastest temperature drop within the shortest amount of time, once the peak temperature within a stack reaches the thermal threshold. In this section, we present a sequence of thread scheduling algorithms.

Since we have two categories of floorplans, we will select a representative homogeneous floorplan as shown in Fig. 1c and a representative heterogeneous floorplan as shown in Fig. 1a. Both the homogeneous and heterogeneous floorplans will be applied with five algorithms: Baseline, Random, Round-Robin, Balancing-by-core, and our proposed Balancing-by-stack algorithms.

4.1 The Baseline

We use the Linux 2.6 scheduler [4] as our baseline algorithm. In this scheduler, each core has a task queue that keeps track of all running tasks on that core. Each queue contains two priority lists: active and expired list. At runtime, the core selects to execute the tasks in the active list, according to some policy. Once a task uses up its time quota, it is moved to the expired list. If all tasks are in the expired list, an epoch has finished, and the scheduler iterates the process by swapping the two lists. Each task in the active list has 10-200 ms of CPU cycle quota, depending on its own priority. By default, the core switches to a different task every 100 ms. Thus, in our 8-core 3D chip, after the scheduling interval of every 100 ms, the scheduler selects a task from each core's active list, according to its original policy, and then assigns it to a different randomly selected core.

This algorithm is simple and has low context switch overhead compared to other algorithms introduced later. However, it may run into the risk of putting two hot tasks into the same core stack, which may lead to extremely high temperature that results in long- and harsh-voltage/frequency scaling penalty to both tasks. Moreover, once a poor scheduling has been made, it stays in that condition for a long period of time (100 ms until the next scheduling time), exacerbating the already serious thermal condition within the chip.

4.2 Random (Baseline+)

A quick fix of the baseline scheduler is to increase the scheduling frequency. In the normal Linux OS, any context switch interval between 10 and 200 ms may be used [4]. A minimum of 10 ms is recommended to avoid unnecessary context switch overhead. We used 8 ms as our scheduling interval mainly due to the experimental restriction on collecting the power traces. Also, 8 ms is close to the thermal constant of the core under testing. However, the algorithm can directly be applied to any scheduling interval recommended in Linux, such as 10 ms, if those restrictions do not apply. Further, we take into account the extra context switch overhead using an 8 ms scheduling interval during our experiments. We performed a real machine measurement on the time required to perform a single context switch. For an 8 ms interval, it is approximately 0.44 percent, a mild penalty that can easily be offset by the performance gain from a better scheduling.

With the improved baseline scheduling algorithm (termed Random to reflect the scheduling decision), the chip can exit a poor thermal condition due to an unwise scheduling more quickly, resulting in less harmful impact.

4.3 Round-Robin

The Random scheduler may result in uneven distribution of power and temperature as tasks are assigned randomly to any core. A Round-Robin (RR) scheduler can overcome this by rotating tasks among cores in a fixed order periodically.

Therefore, after N iterations, where N is the number of cores, each task has been executed on every core for one scheduling interval, e.g., 8 ms. This can help balance the power and temperature distribution in the long run.

4.4 Temperature Balancing by Core

An alternative way to balance the heat among the cores is to explicitly arrange the tasks according to their power consumption and core temperatures. Essentially, a high-power task should be assigned to a low-temperature core. At each scheduling point, the scheduler sorts the power consumption of all tasks and the current temperature of each core. It then assigns the task with the highest power to the coolest core, the second highest power to the second coolest core, and so forth.

Such a mechanism should perform a better job in balancing the temperature distribution among cores than RR. However, recall that there is a strong thermal correlation between two adjacent layers and the cores in one stack have only a small difference in temperatures. This implies that if a core stack contains the hottest core, it probably also contains the second hottest core. When the temperature Balancing-by-core algorithm is applied, the tasks with the lowest and the second lowest powers are scheduled to this hot core stack. Similarly, the tasks with the highest and the second highest powers will be scheduled to the coolest core stack. After that, the hottest/coolest core stack will have the largest temperature drop/rise, which may lead to temperature oscillations and task thrashing between those two stacks, potentially leading to more thermal emergencies. In that case, an RR or a Random algorithm may be a better solution.

Another issue with this mechanism is how the power consumption of each task is obtained. Recently, there have been proposals on obtaining the runtime power consumption of an application through probing the *performance counters* in a processor [15]. We also adopt this approach and assume that each core is equipped with such counters that can be used for power estimation. Note that our power estimation need not be very accurate, as we only need the sorted order of the power, not the absolute values.

4.5 Temperature Balancing by Stack

The core-based temperature-balancing algorithm can create thrashing of tasks between the hottest core stack and the coolest core stack, as we analyzed earlier. This is because the algorithm, while trying to balance the temperatures among all cores, treats each core independently. However, as adjacent dies have strong temperature correlations, cores in the same stack should indeed be considered together. Intuitively, we can assume that each stack is a "super" core that has cores with similar temperatures. Hence, scheduling of the tasks within three dimensions can be reduced to scheduling of the "super" tasks within two dimensions. Apparently, a super task is a set of tasks that is assigned to a super core, i.e., a core stack.

4.5.1 Algorithm for Homogeneous Floorplans

We treat homogeneous and heterogeneous floorplans differently in this algorithm as their super cores have different thermal property. We first elaborate on the algorithm for homogeneous floorplan.

Super tasks. Let L be the number of layers in a 3D chip and N be the number of cores per layer. As a super core contains L cores, a super task should also contain L tasks and there are N super tasks. The scheduling of N super tasks among N super cores is now simply a 2D problem, where a balanced temperature distribution is desired. Hence, we first balance the power among super tasks, i.e., let each super task have about the same power, and then balance the temperatures among super cores by scheduling a relatively high-power super task onto a relatively cool super core.

To balance the power among super tasks, we first sort the powers of all $N \times L$ tasks. Let B_{1-N} be N initially empty bins. We will fill powers into these bins such that each bin will contain L tasks and the total powers of each bin are about the same. In descending order of powers, we put each power value into a bin that has the smallest current total power among all bins. This policy attempts to reduce the gap between the smallest and the largest total powers in each step, in order to generate a relatively balanced total power across N bins. Finally, all powers within a bin form a super task. We remark that our policy is only a heuristic as an optimum solution may require an exhaustive search. We aim for a simple, effective, yet low-complexity heuristic because the scheduler makes the decision at runtime.

Task distribution among and within super cores. The goal of producing super tasks is to generate relatively balanced power distribution across super cores. Once the super tasks are formed, we sum up the temperatures of all L cores in a super core and sort them. Similar to the previous procedure, we assign the hottest super core with the super task of the lowest power and so on. Fig. 5 shows example of scheduling eight tasks onto a two-layer, 4-core-per-layer, 3D chip. Steps a-c depict the procedure except for how tasks within a super task are allocated onto different cores within a stack.

As discussed earlier, the top cores are usually hotter than the bottom cores in a core stack. Hence, we should allocate tasks of higher powers onto the bottom cores for better heat removal and tasks of lower powers onto the top cores. For example, if the temperatures of the cores from bottom up are strictly increasing, then the tasks allocated to them should have strictly decreasing powers from bottom up. The last step in Fig. 5 illustrates this policy in a two-layer floorplan.

Scheduling procedure. To sum up, after every scheduling interval (8 ms in our case), the scheduler performs the following steps:

1. Sorts the powers of all tasks. Forms super tasks. Sorts the power sums of the super tasks from *low to high*.
2. For each super core, sums up the temperatures for all cores. Sorts the temperature sums for all super cores from *high to low*.
3. Creates a sequential one-one mapping between the sorted super tasks and sorted super cores.

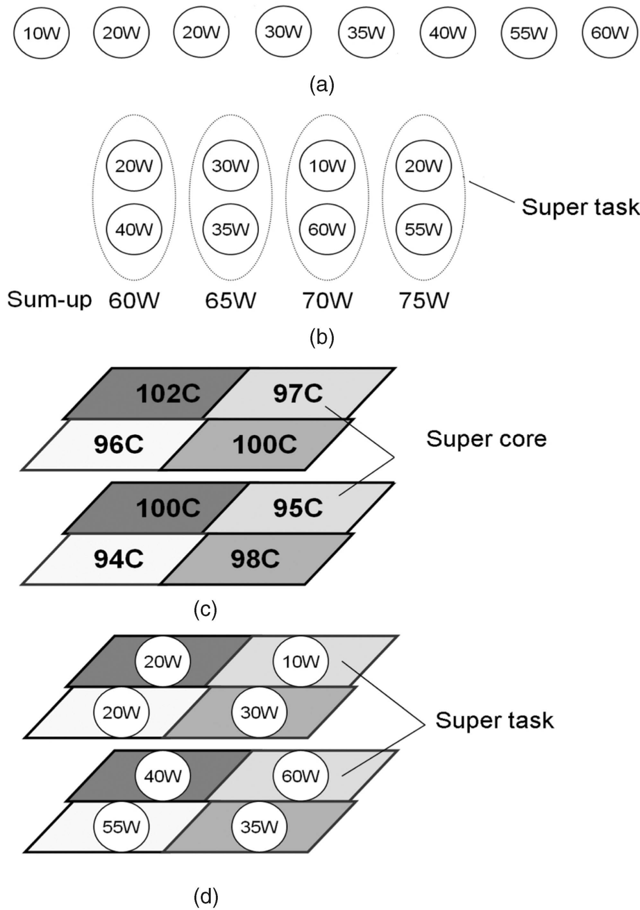


Fig. 5. The temperature balancing-by-stack algorithm. (a) Sort task powers. (b) Combine tasks into super tasks and sort their powers. (c) Sum up the temperatures of super cores and sort them. (d) Assign super tasks onto super cores.

4. In each super core, allocates the tasks in their increasing power order onto the cores with decreasing temperature order.

Our algorithm involves mostly sorting of the powers and temperatures. Therefore, its time complexity is $O(NL\log(NL))$.

4.5.2 Algorithm for Heterogeneous Floorplans

The major difference in heterogeneous floorplans is that different super cores have different heat dissipation capability due to their varying distances to the heat sink. For this reason, even if two super cores are of the same present temperature, the same super task assigned to them will result in different future temperatures. For example, in our experiment for the floorplan shown in Fig. 1a, we assigned eight identical tasks onto eight cores and still observed 4-7 K thermal difference on the top four cores. Therefore, unlike the algorithm for homogeneous floorplans, where the total power among super tasks should be well balanced, the task bundling in heterogeneous floorplan should intentionally create power imbalance to generate balanced temperature distribution among super cores. However, it is difficult to estimate how much power difference we should create among super tasks because the future temperature depends on not only power but also the present temperature and duration of the power. Therefore, for a given set of power values, our algorithm forms the super tasks with minimum, moderate, and maximum total power difference (denoted by Min-diff, Mod-diff, and Max-diff, respectively) and dynamically makes the selection of super tasks.

Let $P_1 \cdots P_n$ be n powers in ascending order. Super tasks with Min-diff, Mod-diff, and Max-diff are formed as follows, assuming each super task contains L tasks:

- Max_diff: $\{P_1, P_2 \cdots P_L\}, \{P_{L+1}, \cdots P_{2L}\}, \cdots$
- Mod_diff: $\{P_1, P_{L+1}, P_{2L+1} \cdots\}, \{P_2, P_{L+2}, \cdots\}, \cdots$
- Min_diff: The principle is to balance the total powers of super tasks. This is identical to the algorithm for homogeneous floorplans (Section 4.5.1).

Intuitively, when the temperature difference among super cores is large, a super task with Max-diff is desired. However, if the power difference among tasks is also large, using the Max-diff may be an overkill. A Mod-diff combination may be sufficient. Therefore, our decision relies on both the temperature gradient (denoted by ΔT) among the super cores and the power range (denoted by ΔP) of the tasks. Let

$$\theta = \frac{\Delta T}{\Delta P}. \tag{4}$$

When θ is small, the temperature gradient (ΔT) is relatively small compared with the power range (ΔP) of the tasks. Super tasks of Min-diff are more appropriate in this situation because we need only to perform mild temperature adjustment. On the other hand, when θ is large, a more aggressive task bundling to create power difference is necessary, hence, the selection favors Max-diff.

During our experiments, we use two heuristic θ values: $\theta_1 = 0.5$ and $\theta_2 = 1$ as the thresholds for choosing different algorithms. The choice of these two values is based on our experimental settings and may vary with thermal properties of the floorplan. If θ falls in the range of $[0, \theta_1)$, Min_diff will be chosen. If θ is in the range of $[\theta_1, \theta_2]$, Mod_diff will be selected. If θ is greater than θ_2, Max_diff will be selected.

4.5.3 A New Thermal Management Scheme

A critical component in company with our proposed scheduling algorithm is how to handle thermal emergencies once a core temperature increases above the hardware threshold. Conventionally, such a core will be put to a low power state through DVFS. In a 3D chip, since the top cores are usually hotter, thermal emergencies usually occur in the top layers. Moreover, our scheduler puts cooler tasks on the top layers, which means that those tasks are more likely to undergo DVFS, leading to their degraded performance.

The problems of such conventional thermal management are twofold. First, the cooler tasks could be penalized more often than the hotter tasks, which bring a fairness issue among different tasks. Intuitively, hotter tasks should be restrained by the system due to their potential harmful impact to the chip. Second, applying DVFS to the cooler tasks on the top layers does not yield the same efficiency as in a 2D chip. This is because it takes longer time to cool down the top cores due to their high-power neighbors at

the bottom. In fact, it is because of those hot bottom tasks are top cores overly heated. Therefore, a more rational thermal management should employ the scalings to the source of an overheating—the bottom cores that are running high-power tasks.

More formally, when core A of a super core S is overheated, the thermal management will select core B with the highest power in S to engage DVFS. Core B may or may not be identical to core A. Such a thermal management strategy solves the two above problems effectively. First, cool tasks are not penalized more often than hot tasks because if a cool task becomes a temperature victim, the hot task that caused the problem will be penalized. Second, all cores in S, including A and B, are quickly cooled because the total power of S is reduced with the maximum strength. For example, in Fig. 5, if the super core containing the 20-40 W super task tripped a thermal emergency on the 20 W core and suppose the DFVS reduced the power of a core by half, then our scheme would reduce the total power of this super core to $20 + 40/2 = 40$ W, while the conventional thermal management will only reduce it to $20/2 + 40 = 50$ W. As we can see, if DVFS is applied to a relatively low-power task, the result is inferior because a task is being penalized, but the total power in the chip is not reduced as much. This is often the case for the temperature Balancing-by-core scheduler as it tends to allocate cool tasks on the top layer (since it is usually hotter).

As a result, our mechanism brings down the temperature of the hotspot at the highest speed, resulting in minimum penalty to the overall performance of this super core. We will later show that our proposed temperature balancing-by-stack scheduling algorithm with improved thermal management results in the much less amount of thermal emergencies and much better performance among all previous schemes.

5 Experimental Methodology

5.1 Floorplan Setup

Our detailed experiments are conducted on floorplans as depicted in Figs. 1a and 1c. Each floorplan has four layers and a total of eight cores. We simulated 8 P4 Northwood cores in 3.0 GHz clock frequency. Each core has a size of $1.144 \times 1.144\ \mathrm{cm}^2$. The remaining space is left for extended cache or memory. So, the die size is $2.289 \times 2.289\ \mathrm{cm}^2$. Other physical parameters such as layer thickness and thermal conductivity of Cu and Si are adopted from Black et al. [3]. For example, the top three layers are thinned to 20 μm while the bulk Si layer closest to the heat sink is of several hundreds of μm.

5.2 Simulation Tool and Power Trace Collection

We used HotSpot [13] version 3.0.2 as our simulation tool. We chose the grid model to experiment our 3D floorplan. We substituted the fourth-order Runge-Kutta method with TILTS [12] to generate accurate temperatures at high speed.

HotSpot takes power traces as inputs and temperature variation within a die is a slower process compared to other metrics such as IPC. Hence, we need to collect extended power traces to model realistic temperature variations such as warming up and cooling down due to the task scheduling. As mentioned earlier, we adopt the recently proposed

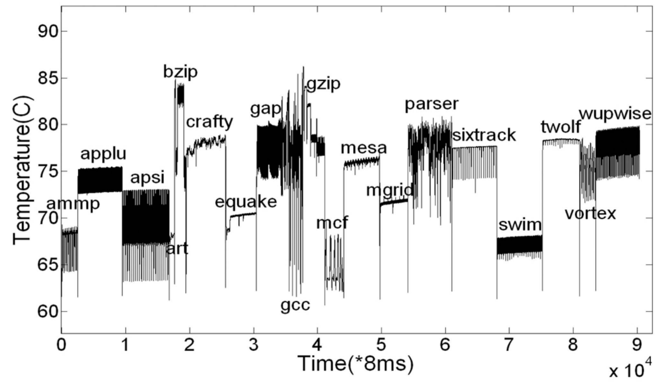


Fig. 6. Temperatures of the benchmark in SPEC2000.

performance counter-based method [15], [25] to collect runtime hardware activities of a program on a real machine. We obtained the power model (calibrated) from [15], [25] to produce long power traces for programs from a Linux machine with a Pentium 4 core. The traces contain powers for each functional unit and all traces are complete execution of the programs in SPEC2K.

For scheduling algorithms that require power information (Balancing-by-core and Balancing-by-stack), we use the power in the last 8 ms interval to predict the power in the next interval. That is, the scheduling decisions are based on local power information. The scheduler needs not to know whether a program is globally hot or cool. Also, we use the last power predictor in the scheduler due to its simplicity. We experimented with more complex power predictors and found that their overhead, both in time and space, is not appropriate for online scheduling [27]. Most of the benchmarks exhibit approximately 5 percent power misprediction rate. Our experiments show that an error within 5 percent makes last power prediction accurate enough for the scheduler.

5.3 Benchmark Classification

We first ran the power traces of each benchmark to obtain its temperature profile, as shown in Fig. 6. We next classified these benchmarks as hot (power intensive), cool (power nonintensive), and mild (between hot and cool). After that, we created nine workload combinations, as listed in Table 1, each with one or more hot tasks. The workload mixes without hot tasks are less thermally critical, and thus, are not considered here. In Table 1, when the number of benchmarks in one combination is less than 8, copies of the benchmarks will be created to ensure that every core in the floorplan has one task to run. This resembles the situation of running parallel threads of the same program in multicore processors.

5.4 DVFS Implementation and Context Switching Overhead

We modified HotSpot to incorporate the hardware DVFS. Every 80 μs, $1/100$ of a scheduling interval, HotSpot checks if the temperature has trespassed the threshold. If so, the voltage is lowered from 1.3 to 1.1 V and the frequency is reduced by $4/5$. We charge 30 μs of overhead on every voltage/frequency transition. During a DVFS scaling, if the temperature persists above the threshold after one 80 μs, the scaling continues and no additional

TABLE 1
The Combination of Benchmarks in Simulation

HC	crafty mcf
HC	sixtrack swim
HHCC	bzip twolf art ammp
HMMC	wupwise equake applu ammp
HM	gzip mgrid
HM	parser equake
HHMM	crafty gzip mgrid apsi
HHMMMCCC	gap twolf equake mgrid vortex ammp art swim
HHHHCCCC	bzip gzip sixtrack wupwise ammp art mcf swim

DVFS switch overhead is charged. We do not choose multilevel DVFS scheme to avoid unnecessary switch overhead in every level transition.

Other overheads in our proposed scheduler are mainly the increased number of context switches. We measured this time in a Linux machine by enforcing a large number of context switches between two tasks and calculated the average switch time from the increased execution time of these two tasks. Such measurement includes also the cache warm-up time required by the tasks. This quantity in our test machine is $\sim 35\ \mu s$. Later, we will see that our proposed scheduler can still outperform Linux baseline scheduler even with much higher context switch frequency.

We set the thermal threshold to trigger DVFS at 105°C. This threshold introduces 6-25 percent (12.4 percent on average) of thermal emergencies, which account for 4-16 percent (8.4 percent on average) of performance degradation under Linux baseline algorithm. Note that the thermal intensity of applications is a feature relative to the emergency threshold. For example, if the average temperature is close/far to/from the threshold, then this application is considered hot/cool. Hence, testing a high threshold would make most programs "cool," and scheduling cool threads is less than necessary. Testing on an overly low threshold would make most programs "hot," which is unrealistic and scheduling would not help anyway. Therefore, we chose 105°C to present practical scenarios and give reasonable room for scheduling threads of different thermal intensity.

6 Results and Analysis

The metrics we use to evaluate different scheduling algorithms are peak temperature of all cores, the reduction in time that a task stays above the thermal threshold (termed "thermal emergency reduction" in later discussion), and performance improvement in terms of total execution time reduction of all tasks. The peak temperature indicates how well a scheduler can alleviate the worst cases of the thermal condition on-chip. The thermal emergency reduction indicates the capability of a scheduler to control the temperature below the hardware threshold. The performance improvement is the result of both the thermal emergency reduction and efficiency of lowering the temperature during an emergency. Next, we present the results for homogeneous and heterogeneous floorplans separately.

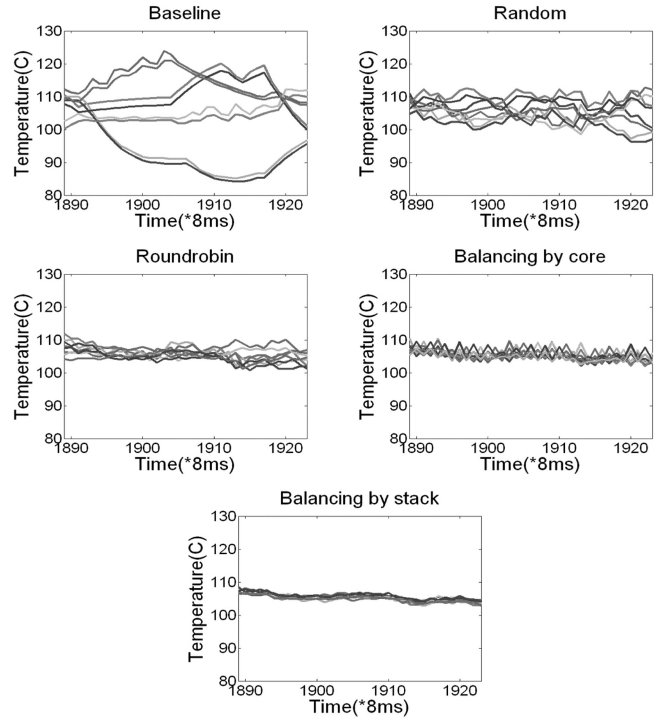


Fig. 7. A zoom-in of temperature variation over time under different scheduling algorithms.

6.1 Homogeneous Floorplan

In this section, we will introduce the experiment results on the thermally homogeneous floorplan. Five schedulers, Baseline, Random, Round-Robin, Balancing-by-core, Balancing-by-stack, are tested in the experiments.

6.1.1 Thermal Behavior Comparison of Different Schedulers

First, let us see a qualitative comparison among different schedulers on the homogeneous floorplan. Fig. 7 shows a closeup of temperature traces for eight cores running the HMMC workload under different scheduling algorithms. Here, we did not enforce DVFS at the threshold because many high-temperature curves would be capped at the threshold. As we can see, the baseline algorithm can result in a large temperature gradient across different core stacks. An approximately 34°C difference between the hottest and the coolest core stacks is observed in this figure. For Random and RR schedulers, the temperature gradient within the 3D chip gradually reduces because their scheduling interval is 8 ms, much smaller than that in the baseline. The temperature gradient is between 4 and 19°C in these schedulers. Finally, both the Balancing-by-core and our proposed Balancing-by-stack schedulers create the smallest temperature gradient among all cores. The temperature curves of all cores almost overlap entirely. The width of the temperature band is 2-6°C only, indicating an excellent balance of temperature among the cores. However, the Balancing-by-core scheduler generates more fluctuation. Note that an ideal temperature balancer would create a 0°C among all the cores. Hence, our proposed Balancing-by-stack algorithm is only a couple of degrees from the ideal case.

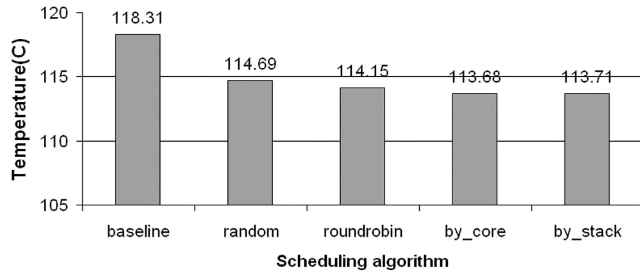


Fig. 8. Peak temperatures of different scheduling algorithms.

6.1.2 Peak Temperature Reduction

Balancing the temperatures across the chip can help reduce the peak temperatures among all the cores. Fig. 8 shows the peak temperature generated from each scheduling algorithm assuming that there are no DVFS employed (otherwise, the peak temperature is just the thermal threshold). We can see in the figures that baseline algorithm can generate the highest peak temperature of 118.31°C. The Random, RR, Balancing-by-core, and Balancing-by-stack algorithms can reduce the peak temperature better and better. Our proposed Balancing-by-stack scheduler generates the second lowest peak temperature of 113.71°C, 4.6°C lower than the baseline and a mere 0.03°C higher than that of the Balancing-by-core.

6.1.3 Thermal Emergency Reduction

A direct benefit from scheduling the tasks is the reduced thermal emergency time, i.e., the time a core temperature stays above the hardware thermal threshold. Note that this metric does not necessarily correlate with the peak temperatures reported in Fig. 8, which are collected under *no DVFS*. For example, a relatively low peak temperature may still trip DVFS if the temperature oscillates around the threshold often. Fig. 9 shows thermal emergency time reductions from different algorithms, normalized to the baseline case. As we can see, the Random, RR, and Balancing-by-core algorithms can reduce the emergency time by 30.9, 37.41, and 36.4 percent, on average, respectively. Our Balancing-by-stack algorithm removes the most emergency time in eight cases of nine benchmarks. An average of 46.23 percent reduction is observed with a range of 6.06-96.04 percent. Also, the Balancing-by-core algorithm turns out to introduce as much emergency time as RR algorithm even with lower peak temperature. This is because 1) it tends to create temperature oscillations among core stacks as discussed in Section 4.4 and 2) it tends to allocate cooler tasks on the top layer, where DVFS is usually engaged for a long time. The consequence is that the overall power in the entire chip is not reduced as much as in other schedulers, where high-power tasks can be scaled during emergency. Therefore, a Balancing-by-core scheduler may not be a good scheduling candidate in practice.

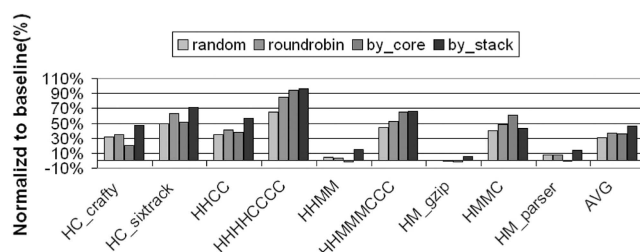


Fig. 9. Thermal emergency time reductions in homogeneous floorplans.

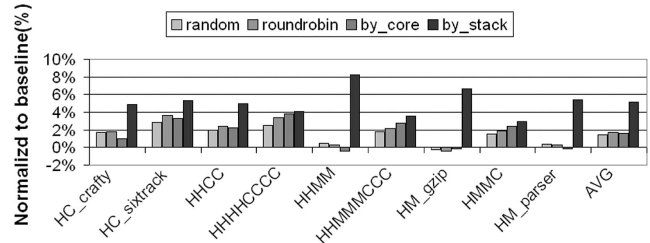


Fig. 10. Performance improvements for homogeneous floorplans.

6.1.4 Performance Improvement

Corresponding to the thermal emergencies removed, our proposed Balancing-by-stack algorithm achieves the best performance improvement among all the algorithms discussed. This is shown in Fig. 10. The performance is the total execution time of all eight tasks in a workload. The results are normalized to the baseline performance. On average, the Balancing-by-stack algorithm achieves a 5.11 percent improvement, while the Random, RR, and Balancing-by-core algorithms achieve 1.45, 1.72, and 1.65 percent improvement, respectively. This is primarily due to the amount of thermal emergencies our algorithm removed, as well as the high efficiency in handling them with our new thermal management mechanism.

We also notice that in some occasions, the performance may not improve even if the thermal emergency time is reduced. This could happen when the temperature floats around the thermal threshold, but does not increase overly high. In such a scenario, there could be many DVFS triggered, which introduce high transition penalty and overkill the gains from scheduling. For example, in the HMMC workload, the Balancing-by-core removed 18.01 percent of thermal emergency time in the Balancing-by-stack, but its performance is 0.54 percent worse than the Balancing-by-stack. Our Balancing-by-stack scheduler removes more thermal emergency time than other schedulers in eight cases of the nine benchmark combinations, and therefore, achieves the most performance improvement.

6.2 Heterogeneous Floorplan

In addition to the five algorithms applied to the homogeneous floorplan, two additional algorithms are also tested for heterogeneous floorplans. The first is the revised Balancing-by-stack algorithm with dynamic super task forming mechanisms. The algorithm is designed to tackle the thermal heterogeneity of the floorplan as discussed in Section 4.5.2. The second is a pseudo-optimal algorithm that tests the quality of each discussed algorithm. We term this algorithm a "1-step-optimal" since it tries all task bundling mechanisms and chooses the one that triggers the fewest DTMs in *one next step*. Note that this is not a true optimal

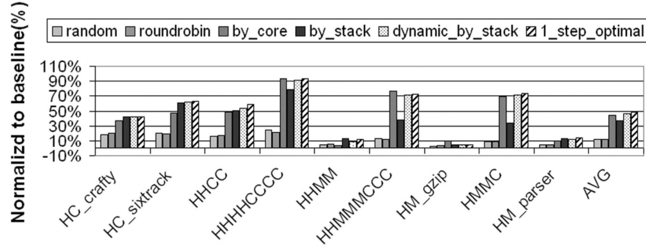


Fig. 11. Thermal emergency time reductions in heterogeneous floorplans.

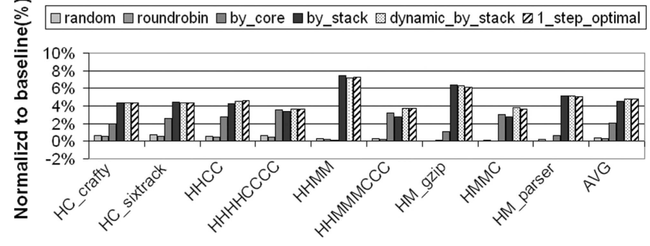


Fig. 12. Performance improvements for heterogeneous floorplan.

algorithm, which would go beyond one-step to enumerate all possible schedules and pick the optimum one (and so is termed "1-step" only). Although it is not realistic to adopt "1-step-optimal" algorithm online due to its complexity, it does indicate the potential for improvement of the discussed algorithms.

6.2.1 Thermal Emergency Reduction

Fig. 11 shows the thermal emergency time reduction for different algorithms normalized to the baseline case. As we can see, Random and RR algorithms perform relatively poorly compared with other algorithms because of the heterogeneity in the floorplan. They achieve 12.41 and 12.35 percent of thermal emergency time reduction, respectively. Our proposed dynamic Balancing-by-stack algorithm achieves a total of 46.37 percent reduction, only 1.92 percent away from the 1-step-optimal, on average, and is better than the remaining algorithms. For example, it removes 9.22 percent more emergency time than the original Balancing-by-stack algorithm. This indicates that dynamically tuning of the task bundling is very helpful to a thermally heterogeneous floorplan. The Balancing-by-core algorithm is slightly better than dynamic Balancing-by-stack in three cases: HHHHCCCC, HHMMMCCC, and HM_gzip. This is because when ΔT and ΔP do not change, our dynamic Balancing-by-stack algorithm will select the same one from the fixed power bundling schemes, while a slight reordering of core temperature will cause Balancing-by-core to form a different and better power bundle more flexibly. Also, Balancing-by-core slightly surpasses 1-step-optimal in HHMMMCCC and HM_gzip workloads. This is because the 1-step-optimal does not generate a global optimal schedule.

6.2.2 Performance Improvement

Compared with the thermal emergencies removed, our dynamic Balancing-by-stack algorithm achieves the best performance improvements, on average, among all the algorithms except the 1-step-optimal.

Fig. 12 shows that Random and RR achieve 0.39 and 0.31 percent improvement, respectively, which is notably lower than 1.45 and 1.72 percent improvement shown in Fig. 10, indicating that Random and RR are not as helpful in heterogeneous floorplans as in homogeneous ones. Balancing-by-stack achieves 2.46 percent improvement more than Balancing-by-core, though Fig. 11 shows that it removes 6.74 percent less thermal emergency time than Balancing-by-core. The reason behind this is that Balancing-by-core tends to generate many overheads in DTM mode switches though the total *time* above the emergency threshold is low, which was reported in Fig. 11. Finally, the dynamic Balancing-by-stack algorithm achieves the best performance improvement of 4.78 percent with negligible gap from the 1-step-optimal.

7 Conclusions

We have demonstrated in this paper that OS task scheduling is an effective approach to improve the thermal conditions in 3D chip multiprocessors. It can reduce the peak temperature within the chip, reduce the thermal emergency triggering amount, and improve the overall performance of the executing tasks.

In particular, we have shown that our proposed scheduling mechanism, Balancing-by-stack, outperforms other intuitive algorithms in the thermally homogeneous floorplan because of the following three properties. First, our scheduler takes into account the high thermal correlations among the layers in one-core stack and schedules tasks in bundles. Second, within every stack of cores, hot tasks are allocated to the layers that are closest to the heat sink for best heat dissipation. Third, upon a thermal emergency, power scaling is engaged in a core stack whose temperature exceeds the threshold, and to the core that generates the largest power in this stack. This can quickly cool down the core stack, reducing the performance penalty imposed to the task.

We also presented a revised Balancing-by-stack algorithm for heterogeneous 3D floorplans. This algorithm-refined task bundling mechanism to adapt to the heterogeneity in thermal distribution among the cores. Such a refinement shows its strength over other algorithms in both thermal emergency reduction and performance improvement. Moreover, it achieves nearly the full potential suggested by a local optimal algorithm.

Acknowledgments

The authors would like to thank Professor Xuandong Li of Nanjing University for his inputs and the anonymous reviewers for their constructive comments. This project is supported in part by the US National Science Foundation (NSF) CAREER 0747242, US NSF CAREER 0641177, NSF-CNS 0720595, NSF-CCF 0734339, and Intel Corp.

References

[1] M. Swarthy and R. Balasubramonian, "Exploring the Design Space for 3D Clustered Architectures," *Proc. Third IBM Watson Conf. Interaction between Architecture, Circuits, and Compilers ($P = ac2$),* Oct. 2006.

[2] K. Banerjee, S. Souri, P. Kapur, and K. Saraswat, "3D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration," *Proc. IEEE,* vol. 89, no. 5, pp. 602-633, May 2001.

[3] B. Black et al., "Die Stacking (3D) Microarchitecture," *Proc. IEEE Int'l Symp. Microarchitecture (MICRO 2006),* pp. 469-479, 2006.

[4] D. Bovet and M. Cesati, *Understanding the Linux Kernel,* third ed. O'Reilly Publishers, Nov. 2005.

[5] J. Choi, C.Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-Aware Task Scheduling at the System Software Level," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '07),* pp. 213-218, Aug. 2007.

[6] S. Chong, L. Shang, and R.P. Dick, "Three-Dimensional Multi-Processor System-on-Chip Thermal Optimization," *Proc. Int'l Conf. Hardware/Software Codesign and System Synthesis,* Sept. 2007.

[7] Y. Deng and W.P. Maly, "2.5-Dimensional VLSI System Integration," *IEEE Trans. Very Large Scale Integration Systems,* vol. 13, no. 6, pp. 668-677, June 2005.

[8] D. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," *Proc. Int'l Symp. Computer Architecture (ISCA),* pp. 78-88, 2006.

[9] B. Goplen and S.S. Sapatnekar, "Thermal via Placement in 3D ICs," *Proc. Int'l Symp. Physical Design (ISPD),* pp. 167-174, 2005.

[10] B. Goplen and S.S. Sapatnekar, "Placement of Thermal Vias in 3D ICs Using Various Thermal Objectives," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 4, pp. 692-709, Apr. 2006.

[11] Y. Han, I. Koren, and C.M. Krishna, "Temptor: A Lightweight Runtime Temperature Monitoring Tool Using Performance Counters," *Proc. Third Workshop Temperature-Aware Computer Systems,* Held in conjunction with ISCA-33, 2006.

[12] Y. Han, I. Koren, and C.M. Krishna, "TILTS: A Fast Architectural-Level Transient Thermal Simulation Method," *J. Low Power Electronics,* vol. 3, no. 1, pp. 13-21, 2007.

[13] W. Huang, M.R. Stan, K. Skadron, K. Sankaranarayanan, and S. Ghosh, "HotSpot: A Compact Thermal Modeling Method for CMOS VLSI Systems," *IEEE Trans. Very Large Scale Integration Systems,* vol. 14, no. 5, pp. 501-513, May 2006.

[14] W. Hung, G.M. Link, Y. Xie, V. Narayanan, and M.J. Irwin, "Interconnect and Thermal-Aware Floorplanning for 3D Microprocessors," *Proc. Seventh Int'l Symp. Quality Electronic Design (ISQED),* pp. 98-104, 2006.

[15] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," *Proc. Int'l Symp. Microarchitecture (MICRO),* pp. 93-104, 2003.

[16] J. Joyner, P. Zarkesh-Ha, and J. Meindl, "A Stochastic Global Net-Length Distribution for a Three-Dimensional System on Chip (3D-SoC)," *Proc. 14th IEEE Int'l ASIC/SOC Conf.,* 2001.

[17] A. Kumar, L. Shang, L.S. Peh, and N. Jha, "HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management," *Proc. Design Automation Conf. (DAC),* pp. 548-553, 2006.

[18] E. Kursun, C.Y. Cher, A. Buyuktosunoglu, and P. Bose, "Investigating the Effects of Task Scheduling on Thermal Behavior," *Proc. Third Workshop Temperature-Aware Computer Systems,* 2006.

[19] G.H. Loh, Y. Xie, and B. Black, "Processor Design in 3D Die-Stacking Technologies," *IEEE Micro,* vol. 27, no. 3, pp. 31-48, May/June 2007.

[20] S. Mysore, B. Agrawal, N. Srivastava, S. Lin, K. Banerjee, and T. Sherwood, "Introspective 3D Chips," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS),* pp. 264-273, 2006.

[21] M.D. Powell, M. Gomaa, and T.N. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density through the Operating System," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS),* pp. 260-270, 2004.

[22] K. Puttaswamy and G.H. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," *Proc. ACM Great Lakes Symp. Very-Large-Scale Integration,* pp. 19-24, 2006.

[23] K. Puttaswamy and G.H. Loh, "Thermal Herding: Microarchitecture Techniques for Controlling HotSpots in High-Performance 3D Integrated Processors," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* pp. 193-204, 2007.

[24] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M.R. Stan, and W. Huang, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Trans. Architecture and Code Optimization,* vol. 1, no. 1, pp. 94-125, Mar. 2004.

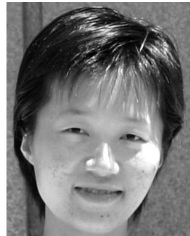
[25] W. Wu, L. Jin, J. Yang, P. Liu, and S.X. Tan, "A Systematic Method for Functional Unit Power Estimation in Microprocessors," *Proc. Design Automation Conf. (DAC),* pp. 554-557, 2006.

[26] Y. Xie, G. Loh, B. Black, and K. Bernstein, "Design Space Exploration for 3D Architecture," *ACM J. Emerging Technologies for Computer Systems,* vol. 2. no. 2, pp. 65-103, Apr. 2006.

[27] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic Thermal Management through Task Scheduling," *Proc. Int'l Symp. Performance Analysis of Systems (ISPASS),* pp. 191-201, 2008.

[28] X. Zhou, Y. Xu, Y. Du, Y. Zhang, and J. Yang, "Thermal Management for 3D Processor via Task Scheduling," *Proc. Int'l Conf. Parallel Processing,* Sept. 2008.

Xiuyi Zhou received the BS and MS degrees in computer science from Nanjing University, China, in 2002 and 2005, respectively. He is currently working toward the PhD degree in electrical engineering at the University of Pittsburgh. His research interests are mainly involved with thermal management for single-core processor and multicore processor. He is also interested in power and thermal management for processors with process variation issues.



Jun Yang received the PhD degree in CS from the University of Arizona in 2002. She is an assistant professor of electrical and computer engineering at the University of Pittsburgh. From 2002 to 2006, she was an assistant professor of computer science and engineering at the University of California, Riverside. Her research interests include low power microprocessor design, thermal management, and 3D chip integration. She is a recipient of the US National Science Foundation (NSF) CAREER Award in 2008. She is a member of the IEEE.

Yi Xu received the BS degree in physics from Nanjing University, Nanjing, China, in 2004, and the MS degree in microelectronics in 2007. She is currently working toward the PhD degree in electrical and computer engineering at the University of Pittsburgh, Pittsburgh. Her research interests include the areas of interconnect architecture design for 2D/3D Chip Multiprocessor (CMP), thermal management for 3D Processors, and the design of digital integrated circuits. She is a student member of the IEEE.

Youtao Zhang received the PhD degree in computer science from the University of Arizona in 2002. He is an assistant professor in the Computer Science Department, University of Pittsburgh. His research interests are in the areas of compilers, computer architecture, and system security. He is a recipient of the US National Science Foundation (NSF) CAREER Award in 2005, the Distinguished Paper Award of the IEEE/ACM International Conference Software Engineering (ICSE) conference in 2003, and the most Original Paper Award of the International Conference Parallel Processing (ICPP) conference in 2003. He is a member of the ACM and the IEEE.

Jianhua Zhao received the PhD degree from Nanjing University in 1999. He is currently a professor in the Department of Computer Science and Technology at Nanjing University. His research interests include software engineering and formal method, especially model checking technique for real-time systems. He has directed or anticipated several Chinese National Natural Science Foundation projects and Chinese Hi-Tech projects. He has published more than 20 research papers in journals and international conferences. He is a member of the IEEE.

▹ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.