Thermal Management for 3D Processors via Task Scheduling *

Xiuyi Zhou[†] **Yi Xu**[†] **Yu Du**[‡] **Youtao Zhang**[‡] **Jun Yang**[†]
[†]**Electrical and Computer Engineering**
University of Pittsburgh, Pittsburgh PA 15261
[‡]**Computer Science**
University of Pittsburgh, Pittsburgh PA 15261

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 across the dies. However, a major challenge in 3D technology is the increased power density which brings 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. Our proposed algorithm can remove on average 54% of hardware DTMs and result in 7.2% performance improvement over the base case.

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. The 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 [15]. 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$\sim$20°C in a two-layer 3D implementation for an Alpha-like processor, compared to a 2D design [13, 21]. Other studies on logic-logic stacking 3D floorplans [1, 3, 22] also show similar thermal constraint.

There are existing 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 leads 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, 16, 17, 20]. OS-assisted task scheduling can reduce the number of times DTMs are triggered while still meeting the thermal constraint. This technique 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, 25]. 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 are 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 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 54% of hardware DTMs and obtain a speedup of 7.2% over the baseline.

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 this paper.

2 Prior Work

There have been many works recently investigating the performance potential and the challenges in 3D CMP designs. Mysore et al. [19] proposed to stack on top of a normal

*This work is supported in part by NSF grants CCF-0734339, CNS-0720595, CAREER 0747242, and CAREER 0641177.

0190-3918/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPP.2008.51

IEEE computer society

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. [25] reported that the peak temperature in a 3D chip of 2 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 et al. proposed a "Thermal Herding" design [22] 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 et al. [9] studied that propoer placement of thermal vias in 3D IC design can obtain a maximum of 47.1% reduction in temperature. In the multicore domain, Loh et al. [18] introduced different approaches for implementing single-core and multicore 3D processors. Particularly, they pointed out that stacking seperate 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 [16] technique controls temperature by limiting the execution of a hot job once it enters an alarm zone. This is achieved by giving hot jobs fewer timeslices and giving cool jobs more timeslices to execute. An ideal simulation study was performed in [17] 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, Choi et al. [5] compared and implemented three different task schedulers, heat-balancing, deferred execution, and threading with cool-loops, to leverage temporal and spatial heat slacks among application threads. The proposed mechanisms are implemented in PowerPC5. 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 to slowdown the workloads as long as the deadlines are met is quite different from our approach focusing on best performance and low thermal profile.

3 Motivation and Rationale

3.1 Floorplan choices

There have been a number of 3D CMP floorplans, as shown in Figure 1 (a)-(c), proposed in literature [1, 3, 18]. One observation we make is that for a 3D stacked chip to be scalable in layer count, it is inevitable to encounter more than one active cores in one vertical core column, no matter how the floorplan staggers them with cache banks (see Figure 1 (a)and (b)). Also, for core layers and cache layers that are "sandwiched" as shown in (c), the cache layers almost serve as heat conductance between the core layers. The heat from any core can quickly propagate vertically to other cores above and below. Extracting the commonality among various 3D floorplans and projecting the future trend in integrating more layers, we choose to use the floorplan in Figure 1(d) as a representative and also a more difficult case than earlier floorplans. Here, there are two layers, and each layer contains four cores. The cache banks are subsumed within each core. Our results therefore serve as the lower bound of the efficiency in thermal-aware task scheduling.

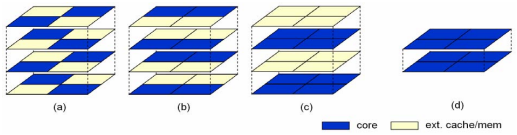

Figure 1. 3D chip multiprocessor floorplan options.

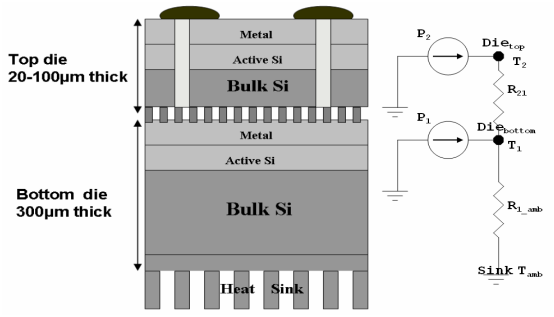

Figure 2. A face-to-back 3D die stacking structure (adapted from [3]), and the corresponding thermal model.

3.2 Vertically adjacent layers have strong thermal correlations

Similar to a regular 2D processor where heat dissipates mostly in the vertical direction [12], 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. Figure 2 shows a basic two-layer 3D chip structure (adapted from [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 ($< 100\mu m$). This directly determines the high heat conductivity between the two adjacent dies. The heat transfer model for this 3D chip is shown on the right of the figure.

Here one die is modeled using one node. Its temperature and power are denoted as 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 temperature (relative to the ambient air) in the bottom and top node respectively. Then,

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

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

Hence, the temperature difference between the two nodes is $R_{21}P_2$. From the parameter used in literature [3, 7, 12, 23], R_{21} is $0.0108{-}0.0159K/W$. P_2 represents the power generated by the entire die. This value is in the range of $40-70W$ for a typical single-core processor. Therefore, the temperature difference between the top and bottom die is merely a $0.43-1.11K$.

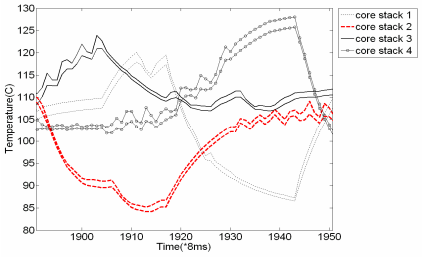

Figure 3. Thermal correlation between adjacent dies.

Such a strong thermal correlation between the two adjacent dies can also be demonstrated from our simulation. Figure 3 shows a typical thermal profile of running eight threads concurrently on eight cores as floorplaned in Figure 1 (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 from Figure 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. 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 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 4-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 further 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, 18]. For clarity, we refer to the cores further from the heat sink as "top" cores, as illustrated in Figure 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.

Suppose in the thermal model depicted in Figure 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 $40-70W$. C_2 represents how quickly temperature changes from the top die. For a thin die within $100\mu m$ in a 2-layer 3D chip, the thermal capacitance is reported as $23.6-37.4mW{\cdot}s/K$ [3, 23]. dT_2/dt is the temperature change rate within a short time. From our 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 $8ms$ window using Hotspot 3.0.2 for 3D chips. Hence, the right hand side of equation 3 is usually positive with a range of $12-52.3W$. Therefore, T_2 is usually higher than T_1.

We also performed simulations to testify the above observation. We intentionally put the coolest job (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 Figure 4. We can see that the top core has higher temperatures than the bottom layer almost always.

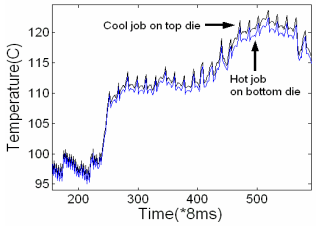

Figure 4. Demonstration of the top die being hotter than the bottom die.

4 Scheduling algorithms

The strong correlations among the cores in one stack leads 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, starting from the simplest baseline algorithm to our proposed algorithm.

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. Once a task uses up its time quota, it is moved to the expired list. If all active 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 $10 - 200ms$ of CPU cycle quota, depending on its own priority. By default, the core switches to a different task every $100ms$. Thus, in our 8-core 3D chip, upon the scheduling interval of every $100ms$, 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 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 ($100ms$ until the next scheduling time).

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 $10 - 200ms$ may be used [4]. A minimum of $10ms$ is recommended to avoid unnecessary context switch overhead. We used $8ms$ as our scheduling interval mainly due to restrictions in collecting the power traces using performance counters. Also, $8ms$ is close to the thermal constant of the core under testing. However, the algorithm can be directly applied to any scheduling interval recommended in Linux such as $10ms$ if those restrictions do not apply. Further, we take into account the extra context switch overhead using an $8ms$ scheduling interval during our experiments. We performed a real machine measurement on the time required to perform a single context switch. For an $8ms$ interval, it is $\sim 0.44\%$, a mild penalty that can be easily 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 scheduler (RR) can overcome this by rotating tasks among cores in a fixed order every $8ms$. This can help balancing 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 the 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 2^{nd} highest power to the 2^{nd} 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 2^{nd} hottest core. When the temperature balancing-by-core algorithm is applied, the tasks with the lowest and 2^{nd} lowest power are scheduled to this hot core stack. Similarly, the tasks with the highest and 2^{nd} highest power 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, a 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 has been proposals on obtaining the runtime power consumption of an application through probing the performance counters in a processor [14]. 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, 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 "super" tasks within two dimensions. Apparently, a super task is a set of tasks that are assigned to a super core, i.e., a core stack.

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 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 power 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, 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. Figure 5 shows an example of scheduling 8 tasks onto a 2-layer, 4-core-per-layer, 3D chip. Step (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. 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. Figure 5's last step illustrates this policy in a two-layer floorplan

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

1. Sort the powers of all tasks. Form super tasks. Sort the power sums of the super tasks from low to high.
2. For each super core, sum up the temperatures for all cores. Sort the temperature sums for all super cores from high to low.
3. Create a sequential one-one mapping between the sorted super tasks and sorted super cores.
4. In each super core, allocate the tasks in their increasing power order onto the cores with decreasing temperature order.

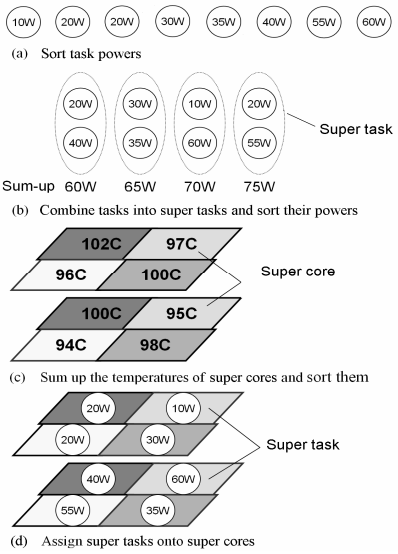

Figure 5. The temperature balancing-by-stack algorithm.

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

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.

The problems of such conventional thermal management are twofold. First, the cooler tasks could be penalized more often than the hotter tasks. For fairness, 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. 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. B may or may not be identical to 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 is 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 Figure 5, if the super core containing the 20W-40W super task tripped a thermal emergency on the 20W core, and suppose the DFVS reduces the power of a core by half, then our scheme will reduce the total power of this super core to $20 + 40/2 = 40W$, while the conventional thermal management will only reduce it to $20/2 + 40 = 50W$. 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.

5 Experimental Methodology

5.1 Floorplan setup

As discussed in Section 3.1, our experiments are conducted on a floorplan as depicted in Figure 1(d). In this floorplan, there are two layers with four cores on each layer. We simulated 8 P4 Northwood cores in 3.0GHz clock frequency. Each P4 core has a size of $1.144 \times 1.144\ cm^2$, so the die size is $2.289 \times 2.289 cm^2$. Other physical parameters of the floorplan are similar to [3].

5.2 Simulation tool and power trace collection

We used Hotspot [12] version 3.0.2 as our simulation tool. We chose the grid model to experiment our 3D floorplan as shown in Figure 1(d). We substituted the 4th-order Runge-Kutta method with TILTS [11] to generate accurate temperatures at high speed.

As mentioned earlier, we adopt the recently proposed performance counter based method [14, 24] to collect runtime hardware activities of a program on a real machine. We obtained the power model (calibrated) from [14, 24] 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 $8ms$ 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 on-line scheduling [26]. Most of the benchmarks exhibit $\sim 5\%$ power mis-prediction rate. Our experiments show that an error within 5% 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 Figure 6. We next classified these benchmarks as hot(power-intensive), cool(power non-intensive), and mild(between hot and cool). After that, we created 9 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.

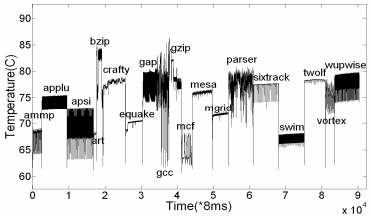

Figure 6. Temperatures of the benchmark in SPEC2000

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

5.4 DVFS implementation and context switching overhead

We modified Hotspot to incorporate the hardware DVFS. Every 80μs, 1/100 of a scheduling interval, Hotspot checks if the temperature has trespassed the threshold. If so, the voltage is lowered from $1.3V$ to $1.1V$ 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 80μs, the scaling continues and no additional DVFS switch overhead is charged. We do not choose multi-level DVFS scheme to avoid unnecessary switch overhead in every level transition.

Other overheads in our proposed scheduler is 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 calculating the average switch time from the increased execution time of these two tasks. This quantity in our test machine is $\sim 35\mu s$.

Finally, our DVFS trigger temperature is set at $108°$C because a 3D chip generates higher temperature than does a planar chip which typically engage DVFS around $80°$C.

6 Results and Analysis

The metrics we use to evaluate scheduling algorithms are peak temperature of cores, the reduction in time that a task stays above the thermal threshold (termed "thermal emergency reduction"), 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 the efficiency of lowering the temperature during an emergency.

6.1 Thermal behavior comparison of different schedulers

First, let us see a qualitative comparison among different schedulers. Figure 7 shows a close-up of temperature traces for 8 cores running the HMMC workload under different scheduling algorithms. Here, we did not enforce DVFS at the threshold because otherwise, 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. A $\sim 35°C$ difference between the hottest and the coolest core stack is observed in this figure. For Random, RR and balacing-by-core scheduler, the temperature gradient within the 3D chip gradually reduces because their scheduling interval is $8ms$, much smaller than that in the baseline. Temperature gradient is between 4-$15°C$ in these schedulers. Finally, our proposed balancing-by-stack creates the smallest temperature gradient among all cores. The temperature curves of all cores almost overlap entirely. The width of the temperature band is $<$ 2-$3°C$ only, indicating an excellent balance of temperature among the cores.

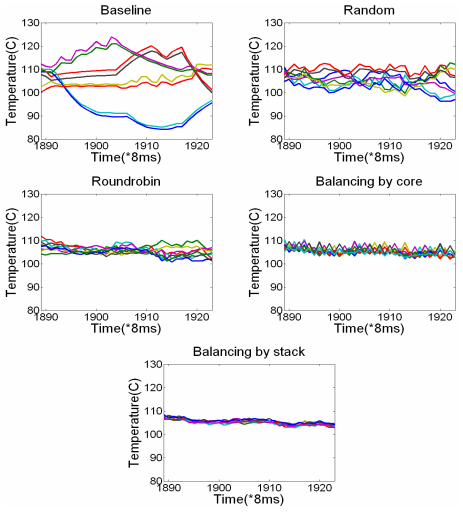

Figure 7. A zoom-in of temperature variation over time under different scheduling algorithms.

6.2 Peak temperature reduction

Balancing the temperatures across the chip can reduce the peak temperatures among all cores. Figure 8 shows the peak temperature generated from each scheduling algorithm assuming there are no DVFS employed. We can see that baseline algorithm can generate the highest peak temperature of $145.08°C$. The Random, RR, balancing-by-core and balancing-by-stack can reduce the peak temperature better and better. Our proposed balancing-by-stack scheduling generates the lowest peak temperature of $121.3°C$, nearly $24°C$ lower than the baseline.

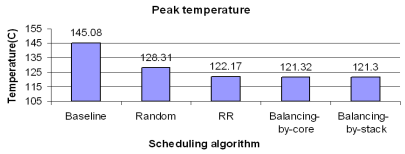

Figure 8. Peak temperatures of different scheduling algorithms.

6.3 Thermal emergency reduction

A direct benefit from the scheduling the tasks is the reduced thermal emergency time. Note that this metric does not necessarily correlate with the peak temperatures reported in Figure 8, which are collected under no DVFS. Figure 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 can reduce the emergency time by 34.45%, 39.61% and 30.14% on average respectively. Our Balancing-by-stack algorithm consistently removes the most emergency time. An average of 53.98% reduction is observed, with a range of 21.37%-82.28%. Also, the Balancing-by-core algorithm turns out to introduce more time in emergency than Random and RR algorithms 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. Therefore the overall power in the entire chip is not reduced as much as in other schedulers where high power tasks can be scaled during emergencies.

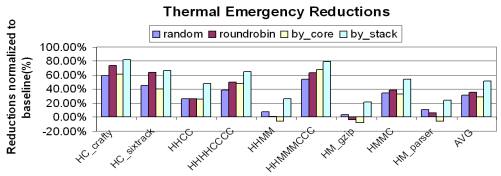

Figure 9. Thermal emergency reductions of different schedulers.

6.4 Performance improvement

Corresponding to the thermal emergencies removed, our proposed Balancing-by-stack algorithm achieves the best performance speedup among all algorithms discussed. This is shown in Figure 10. The performance is the total execution time of all 8 tasks in a workload. The results are normal-

ized to the baseline performance. On average, the Balancing-by-stack achieves a 7.22% speedup, while the Random, RR, and Balancing-by-core algorithm achieve 2.60%, 2.73% and 1.35% 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 mechansim.

In some cases, the performance may not improve even if the thermal emergency time is reduced. This could happen when the temperature floats around the thermal threshold. In such a scenario, there could be many DVFS triggered, which introduce high transition penalty and overkills the gains from scheduling. For example in the HHMM workload, the RR removes 5.14% of thermal emergency time in baseline, but its performance is 0.68% worse than the baseline. Our Balancing-by-stack consistently removes more thermal emergency time than other schedulers and therefore achieves the most performance improvement.

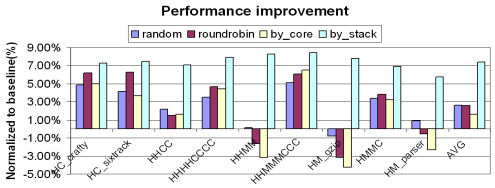

Figure 10. Performance speedups of different scheduler.

7 Conclusions

We have demonstrated in this paper that OS task scheduling is an effective approach to improving 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 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.

References

[1] M. Awasthi, R. Balasubramonian, "Exploring the Design Space for 3D Clustered Architectures", *3rd IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers (P=ac2)*, Yorktown Heights, October 2006.

[2] K. Banerjee, et al., "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration," *Proceedings of the IEEE*, vol. 89, pp. 602–633, May 2001.

[3] B. Black, etc al., "Die Stacking (3D) Microarchitecture," *MICRO 2006*, 469-479

[4] D. Bovet, M. Cesati, "Understanding the Linux kernel, 3rd Edition," *O'Reilly Publisher,* November, 2005.

[5] J. Choi, "Thermal-aware Task Scheduling at the System Software Level," *ISLPED'07*, August 27-29, 2007, Portland, Page 213-218.

[6] S. Chong, L. Shang, and R. P. Dick, "Three-dimensional multiprocessor system-on-chip thermal optimization," *Proc. International Conference Hardware/Software Codesign and System Synthesis*, Sep. 2007

[7] Y. Deng, W. P. Maly, "2.5-dimensional VLSI system integration," *IEEE Trans. VLSI Syst.*, 13(6): 668-677 (2005).

[8] D. Donald, M. Martonosi, "Techniques for multicore thermal management: classification and new exploration," *ISCA*, pp 78-88, 2006.

[9] B. Goplen, et al., "Placement of Thermal Vias in 3-D ICs Using Various Thermal Objectives," *IEEE Trans. on CAD of Integrated Circuits and Systems 25(4)*, pp.692-709 (2006)

[10] Y. Han, et al., "Temptor: A lightweight runtime temperature monitoring tool using performance counters," *the 3rd Workshop on TACS, Held in conjunction with ISCA-33*, 2006.

[11] Y. Han, et al., "TILTS: A Fast Architectural-Level Transient Thermal Simulation Method," *Journal of Low Power Electronics*, 3(1): 13-21 (2007).

[12] W. Huang, et al., "HotSpot: A Compact Thermal Modeling Method for CMOS VLSI Systems," *IEEE Trans. VLSI Syst.*, 14(5):501-513, May 2006.

[13] W. Hung, et al., "Interconnect and Thermal-aware Floorplanning for 3D Microprocessors," *the 7^{th} ISQED*, pp. 98-104, 2006.

[14] C. Isci, M. Martonosi, "Runtime power monitoring in highend processors: methodology and empirical data," *MICRO*, pp. 93-104, 2003.

[15] J. Joyner, et al., "A Stochastic Global Net-length Distribution for a Three-Dimensional System on Chip (3D-SoC)," *the 14^{th} IEEE International ASIC/SOC Conference,* 2001.

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

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

[18] G. H. Loh, Y. Xie, B. Black, "Processor Design in 3D Die-Stacking Technologies," *IEEE Micro*, 27(3): 31-48 (2007).

[19] S. Mysore, et al., "Introspective 3D chips," *ASPLOS 2006*, pp. 264-273.

[20] M. D. Powell, et al., "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," *ASPLOS 2004*, pp. 260-270.

[21] K. Puttaswamy, G. H. Loh "Thermal analysis of a 3D die-stacked high-performance microprocessor," *ACM Great Lakes Symposium on VLSI*, 19-24, 2006.

[22] K. Puttaswamy, et al., "Thermal Herding: Microarchitecture Techniques for Controlling HotSpots in High-Performance 3D-Integrated Processors," *HPCA*, pp. 193-204, 2007.

[23] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M.R. Stan, and W. Huang., "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM TACO*, 1(1):94-125, Mar. 2004.

[24] W. Wu, L. Jin, J. Yang, P. Liu, S. X.-D. Tan "A systematic method for functional unit power estimation in microprocessors," *DAC 2006*, 554-557.

[25] Y. Xie, et al., "Design Space Exploration for 3D Architecture," *ACM Journal of Emerging Technologies for Computer Systems*, Vol. 2. No. 2, pp.65-103, April 2006.

[26] J. Yang, et al., "Dynamic Thermal Management through Task Scheduling," *ISPASS,* pp.191-201, 2008.