Short Papers

On the Thermal Attack in Instruction Caches

Joonho Kong, Johnsy K. John, Eui-Young Chung, Sung Woo Chung, and Jie Hu

Abstract—The instruction cache has been recognized as one of the least hot units in microprocessors, which leaves the instruction cache largely ignored in on-chip thermal management. Consequently, thermal sensors are not allocated near the instruction cache. However, malicious codes can exploit the deficiency in this empirical design and heat up fine-grain localized hotspots in the instruction cache, which might lead to physical damages. In this paper, we show how instruction caches can be thermally attacked by malicious codes and how simple techniques can be utilized to protect instruction caches from the thermal attack.

Index Terms—Cache memories, fine-grain localized hotspot, malicious codes, microprocessors, thermal attack.

---

1 Introduction

As technology trends are packing transistors even more tightly, the on-chip power densities are exponentially increasing [1]. In addition, slow supply voltage scaling further deteriorates this situation. The situations are getting even worse, as the localized power densities grow exponentially with the advanced process technology. Power densities have become high enough to cause serious thermal challenges, possibly even resulting in a project cancellation [36]. This in turn leads to demands for much larger cooling capacity in the microprocessor designs, thus significantly increasing the costs of cooling systems and chip packaging [34]. Without proper consideration of thermal problems, permanent damage (thermal runaway) or gradual damage (accelerated aging) on the microprocessor would be suffered [6].

To overcome the thermal challenges in a cost-effective manner, dynamic thermal management (DTM) techniques are proposed [3], [12], [30], [32]. DTM monitors chip-wide temperature at runtime and dynamically invokes power reduction schemes to avoid thermal emergency once the temperature exceeds a predefined DTM trigger threshold, which allows less expensive packaging/fan solution. In the meantime, thermal hotspots are not spatially static but dynamically moving over time, depending on which on-chip structures (e.g., register file, integer arithmetic, floating-point arithmetic, etc.) are most heavily used. As spatial thermal differences are exponentially increasing with distance [19], a single sensor is not sufficient to track temperature changes across a large chip. Especially, it may become a problem in the case of malicious software that tries to create a hotspot in an unmonitored structure. This is different from the thermal denial-of-service (DOS) attack presented in [6], [11].

Though caches are generally not included in the hottest functional units in microprocessors, they could be targets for thermal attacks, which may lead to thermal runaway in the caches. Against the thermal attack in on-chip caches, there might be several solutions at the circuit-/packaging-level. First one is to increase the number of thermal sensors across the caches. Though more thermal sensors make the microprocessor more reliable, sensors must be fairly large for high accuracy, making them costly in terms of area and power. Furthermore, increasing the number of sensors also increases the testing and calibration cost for each chip. Even with this overhead, IBM employs nine thermal sensors for a core in Power6 microprocessor [8]. However, the thermal sensors are not deployed in relatively cool units such as on-chip caches rather in most likely hotspots, in order to control the incurred cost of thermal sensors. Second solution is to consider the worst case where a hotspot is farthest from a thermal sensor. However, microprocessors designed under this worst case assumption will suffer from severe performance degradation by extremely over-estimating the temperature. According to our simulation, in case of the worst case design, an average performance loss of more than 50 percent is observed when executing the SPEC2000 INT benchmark suite. Moreover, microprocessor designers should consider the thermally malicious code to set appropriate thermal guard bands. In most modern microprocessors, however, they set the thermal guard bands considering the normal application behavior to optimize performance under thermal constraints such as Thermal Design Point (TDP). The trade-off among reliability, performance, and cost, paradoxically, gives the malicious code an opportunity to thermally attack the microprocessor.

The contribution of this paper is summarized as follows: This paper is the first-time introduction to thermal attacks on unexpected fine-grain hotspots and their protection methods. The rest of the paper is organized as follows: Section 2 discusses the related work on temperature measurement, temperature management, and thermal security. Section 3 introduces the possibility of the thermal attack in microprocessors and the potential target of the thermal attacks. Section 4 describes our fine-grain cache modeling. Section 5 presents examples of a malicious code for the thermal attacks. Section 6 proposes simple and effective protection techniques. Section 7 presents evaluation results, and lastly we conclude this paper in Section 8.

2 Related Work

2.1 Temperature Measurement

Conducting thermal analysis in the early stage of microprocessor design is of critical importance to its future success. Skadron et al. developed an architectural-level temperature modeling tool, called HotSpot [13], [31], [32], which enables computer architects to evaluate architectural-level thermal-aware designs. HotSpot is a software model for simulating on-chip temperatures at a micro-architectural granularity. It models the processor as a network of thermal resistors and conductors per functional unit, where power dissipation in each unit is treated as current source in the RC network. The power dissipation for each unit is obtained by the access counts in the architectural simulator. Lee and Skadron extended HotSpot to interface with performance counters in order to get activity data directly from a real processor [20]. They estimated power dissipation from performance counters based on a model developed by Isci and Martonosi [14]. Since most recent

- *J. Kong is with the Division of Computer and Communications Engineering, Science Library, Room 604A, Korea University, Anam-dong 5-ga, Seongbuk-gu, Seoul 136-713, South Korea. E-mail: luisfigo77@korea.ac.kr.*
- *J.K. John is with the AMD Boston Design Center, Boston Design Center, Advanced Micro Devices, Inc., Boxborough, MA 01719. E-mail: johnsy.john@amd.com.*
- *E.-Y. Chung is with School of Electrical and Electronic Engineering, Engineering Bldg #2 B620, Yonsei University, Sinchon-dong, Seodaemun-gu, Seoul 120-749, South Korea. E-mail: eychung@yonsei.ac.kr.*
- *S.W. Chung is with the Division of Computer and Communications Engineering, Science Library, Room 404C, Korea University, Anam-dong 5-ga, Seongbuk-gu, Seoul 136-713, South Korea. E-mail: swchung@korea.ac.kr.*
- *J. Hu is with Department of Electrical and Computer Engineering, Newark College of Engineering, New Jersey Institute of Technology, University Heights, Newark, NJ 07102. E-mail: jhu@njit.edu.*

Manuscript received 17 Apr. 2008; revised 1 Dec. 2008; accepted 5 Mar. 2009; published online 20 Mar. 2009.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2008-04-0075.
Digital Object Identifier no. 10.1109/TDSC.2009.16.

microprocessors, including embedded processors, have performance counters, HotSpot extension using performance counters presents opportunities to measure the localized temperature of real processors. On the other hand, its computation was too heavy to be adopted in real temperature monitoring, though it is a nice temperature modeling tool. Note that complicated differential equations should be solved in order to find localized temperatures in HotSpot. To monitor the localized temperatures with negligible performance overhead, Chung and Skadron proposed a simple formula using simple regression analysis [5].

2.2 Temperature Management

Huang et al. [12] proposed a DVS-based technique for thermal control. Though they investigated the memory hierarchy, they did not examine other hot functional blocks such as register files. Brooks and Martonosi [3] set a constant threshold power and they applied five thermal control techniques (clock frequency scaling, voltage and frequency scaling, decode throttling, speculation control, and I-cache toggling), when the threshold power was exceeded. They found DFS and DVS to be inefficient because of the invocation overhead (more than 10 ms compared to the short sampling period, 10,000 cycles). Skadron et al. [30] proposed a formal control theory for DTM. Different from the previous studies that adopted constant trigger temperature (or power) and fixed response, they allow the fetch-toggling rate to be changed according to the thermal history that may need additional storage. There are some previous works [21], [25] on thermal management in SMP systems, which schedules the tasks to make use of the idle SMP nodes. Srinivasan and Adve proposed the predictive DTM by profiling multimedia applications [33]. All the studies target at the overall chip temperature or the localized temperature whose granularity is a functional unit (e.g., integer register, integer execution unit, issue unit, and L1 instruction cache).

As far as we know, there have been only two studies on more fine-grain temperature management, which investigated the thermal behavior of cache subarrays. To prevent several subarrays from being overheated, John et al. [16] proposed two subarraying schemes: separated subarray and interleaved subarray. To reduce the excessive temperature, which leads to more leakage, Ku et al. [18] maximized the distance of the blocks with consecutive addresses. Both goals are to spread the cache accesses across the subarrays, not to access adjacent subarrays. However, they only concentrated on spreading heat for power and area efficiency, not considering thermal attacks.

2.3 Thermal Security

Dadvar and Skadron [6] studied DTM-related thermal security such as thermal DOS attack and data integrity problems. They mentioned that a malicious code could cause a hardware malfunction or burn-out of a chip, but they did not find an example. Hasan et al. [11] introduced power density induced DOS attack in Simultaneous multithreading (SMT) processors. They showed that malicious thread can degrade the performance of normal threads. Paul et al. [23] introduced thermal attack in storage systems. They revealed that thermal attack is possible on storage system. However, to the best of our knowledge, there has been no study on thermal attack (not DOS) in a microprocessor caused by a real malicious code.

3 Potential Thermal Attack by Malicious Codes

3.1 Processor's Vulnerability to Thermal Attacks

As pointed out in [31], localized heating up is much faster than chip-wide heat transfer that creates thermal hotspots across the entire chip. This requires deploying thermal sensors at each possible hotspot for accurate thermal tracking, which, however, may not be feasible due to the cost and power consumption of these on-chip thermal sensors. Typical compromised solution is to put a limited number of thermal sensors at those known most likely hotspots based on the thermal behavior of the normal applications. Since a limited number of thermal sensors which are deployed in generally known hotspot are not sufficient to track temperature changes across a large chip [10], [19], [22], it also leaves the opportunity for malicious software to create hotspots in unmonitored functional blocks. In modern microprocessor designs, designers make theoretical thermal maps which guide an efficient thermal sensor placement. Using those thermal maps, thermal guard bands are set to guarantee that nowhere in the microprocessor is over the maximum reliable temperature [10]. However, since thermal maps are extracted based on the behavior of normal applications, these thermal guard bands guarantee the reliability only when the normal application is executed. In addition, microprocessor designers usually optimize their DTM techniques for normal applications to minimize performance overheads. In [22], they introduced an efficient and reliable sensor placement. However, they also allocated thermal sensors based on the normal application behavior. Thus, thermal malicious codes could avoid the thermal runaway detection by the thermal sensors. If the theoretical thermal maps do not consider the thermal malicious codes, unmonitored hotspots can physically damage the circuitry without being detected by faraway thermal sensors (recall that performance will be severely degraded, when malicious codes are considered for thermal guard bands as mentioned in Section 1). In other words, thermal guard bands may not be sufficiently large to detect unusual thermal behavior due to a long distance between the thermal sensors and the target of thermal malicious code. In this case, the target of the thermal attack may be heated over the thermal emergency temperature without being detected by faraway thermal sensors. Furthermore, when the thermal attack concentrates on more fine-grain units, it is more difficult to detect the thermal emergency.

3.2 Targets of Thermal Attacks

It is not easy to attack the generally known hotspots because thermal sensors are typically deployed near these hotspots. The L1 caches are largely ignored for thermal protection, since they are not recognized as thermal hotspots when running normal applications. Table 1 shows peak temperatures of the functional blocks when running SPEC2000 benchmarks in our simulated microprocessor. (The detailed specification of our simulated microprocessor is described in Section 7.1.) To examine the thermal behavior of the normal applications, we used SimPoint [27] to gain more exact simulation results. We deploy thermal sensors in nine functional units which are generally known as hotspots. (Functional units that are equipped with thermal sensors are emphasized by gray-shaded, bold, and italic font in Table 1.) Considering the maximum junction temperature in 70 nm technology [29], DTM trigger and emergency temperature is set to $95^{\circ}\mathrm{C}$ and $100^{\circ}\mathrm{C}$, respectively. When the trigger temperature is reached, dynamic voltage and frequency scaling (DVFS) operates to cool down the microprocessor. When the emergency temperature is reached, the microprocessor stops fetching instructions until it is sufficiently cooled down.

As shown in Table 1, the L1 caches are relatively cool units compared to the other units. Moreover, the L1 caches occupy relatively large area in microprocessors, which makes it difficult to detect the fine-grain localized hotspots. Between the data cache and the instruction cache, the data cache seems to be more attractive to attack. Since the data cache can be accessed several times in one cycle due to the superscalar feature. However, effective address calculation of load/store instructions makes the integer register file much hotter than the data cache, which invokes DTM. On the other hand, the L1 instruction cache is not directly related to microprocessor's back-end functional units. Furthermore, it can be accessed in every cycle by well-constructed malicious codes. Therefore, we choose the instruction cache as our target.

TABLE 1
Peak Temperatures of Major Processor Functional Units When Running SPEC2000 Benchmarks

	I$-max	D$-max	L2-left	L2-bottom	L2-right	Bpred	DTB	***FPAdd***	***FPReg***	***FPMul***	***FPMap***	IntMap	***IntQ***	***IntReg***	***IntExec***	***FPQ***	***LdStQ***	ITB
gzip	72.6	68.0	62.5	62.4	62.9	67.1	67.2	64.3	63.6	63.4	63.4	69.2	79.2	97.0	80.1	66.3	73.6	69.1
vpr	74.8	72.3	66.2	66.1	66.8	71.0	71.5	71.6	69.8	68.4	67.9	73.3	82.1	95.7	82.2	71.6	78.6	73.8
gcc	72.8	73.7	63.8	63.7	64.3	68.4	69.1	65.8	65.1	64.8	64.8	70.5	81.2	97.0	81.0	68.2	77.4	71.4
mcf	72.9	71.2	68.1	68.2	68.3	70.6	70.3	68.8	68.6	68.5	68.5	70.7	76.9	85.3	78.2	69.7	73.9	71.3
crafty	69.5	68.1	62.5	62.4	63.0	67.0	67.4	64.4	63.7	63.5	63.5	69.5	80.0	97.0	80.4	66.6	74.5	69.5
parser	73.8	70.7	63.9	63.8	64.3	69.5	68.8	66.2	65.3	64.9	64.8	70.7	80.8	97.1	81.7	68.2	75.9	71.0
eon	71.9	71.9	63.3	63.1	63.7	70.6	68.9	72.4	70.0	68.1	66.2	71.7	82.0	96.7	81.8	70.3	77.9	71.7
perlbmk	70.6	70.4	63.0	62.9	63.5	68.4	68.5	65.3	64.5	64.2	64.2	70.6	81.1	96.9	81.5	67.7	76.2	70.8
gap	71.3	69.9	63.8	63.7	64.4	68.8	69.1	66.0	65.4	65.1	65.1	71.4	81.6	96.7	82.1	68.5	76.4	71.4
vortex	71.3	69.5	62.8	62.7	63.2	69.4	67.9	65.4	64.3	63.8	63.7	69.4	80.1	97.0	81.4	67.2	75.2	70.0
bzip2	71.6	73.1	62.8	62.7	63.1	67.1	67.4	64.3	63.6	63.4	63.4	69.2	80.1	97.3	80.7	66.5	75.4	69.5
twolf	71.9	69.4	63.9	63.7	64.3	69.7	69.0	68.6	67.1	65.8	65.4	71.1	80.6	96.7	81.2	69.0	75.5	71.1
wupwise	74.8	71.3	65.5	65.0	65.6	74.0	70.4	81.3	80.1	78.7	72.7	74.7	81.6	95.1	81.6	75.1	77.8	73.2
swim	73.4	70.6	68.3	68.1	68.2	71.9	70.0	79.3	78.4	77.6	72.3	72.1	74.1	78.9	73.6	73.5	73.0	71.2
mgrid	75.8	71.4	66.9	66.3	66.7	76.6	70.3	95.2	87.0	78.2	73.4	73.7	76.3	82.4	74.5	78.6	76.6	72.9
applu	77.6	73.9	68.1	67.6	67.9	73.2	70.5	83.8	84.3	86.6	75.7	74.1	76.4	82.6	75.0	76.4	76.0	72.2
mesa	73.0	73.5	63.4	63.2	63.7	70.0	68.2	72.4	70.4	68.8	66.3	70.8	80.6	96.8	80.9	69.8	76.1	70.7
galgel	72.0	71.2	63.0	62.8	63.3	68.8	67.9	74.8	75.1	75.9	69.5	71.6	80.6	97.0	79.5	71.0	77.3	70.8
art	74.0	72.5	63.2	63.0	63.5	71.4	68.8	75.1	73.1	71.3	67.4	70.8	79.5	96.9	80.2	70.5	75.2	70.5
equake	70.9	70.8	68.2	68.1	68.3	70.4	69.6	72.8	72.4	72.3	70.4	70.9	73.3	77.7	72.8	71.0	72.6	70.6
facerec	75.2	72.3	65.1	64.7	65.2	72.6	69.9	84.6	80.6	77.5	71.5	73.8	81.4	96.9	81.3	73.8	77.0	72.4
ammp	76.0	71.5	64.1	63.6	64.1	72.9	69.2	81.8	82.2	82.7	73.5	74.1	81.0	96.1	80.0	74.9	78.0	72.2
lucas	74.2	69.8	68.4	68.0	68.1	73.3	69.6	80.9	79.6	78.1	73.8	72.3	71.9	74.3	71.2	74.3	71.9	70.8
fma3d	75.1	74.6	67.7	67.4	67.8	73.8	71.7	82.5	82.2	82.0	74.4	75.2	81.7	93.6	80.6	76.2	79.9	74.3
sixtrack	76.6	68.8	63.9	63.2	63.5	74.7	66.9	92.9	95.2	96.7	80.2	74.0	73.7	80.1	71.9	78.5	73.3	69.6
apsi	76.7	75.1	66.1	65.7	65.9	73.9	70.1	85.6	83.5	81.5	73.7	74.4	81.5	96.9	80.7	75.3	78.0	72.6

4 Fine-Grain Thermal Modeling and Characterization

To enable an accurate thermal analysis of the instruction cache under thermal attacks, we model the thermal behavior of the instruction cache at fine granularity that captures the physical implementation details of Alpha 21364 as close as possible. We utilize the Cacti cache model [26] to derive the detailed geometric sizes and power distribution of the instruction cache at the macroblock level. According to Cacti [26], both the data and tag arrays are divided into subblocks, and further into subarrays. Each subarray is largely an autonomic unit where both the bit-cell array and peripheral circuits are activated during the access to the subarray. For a close look into the internal cache structures, we first breakdown the data subarrays into major functional blocks: bit-cell array, postdecoder, column multiplexer, precharge circuit, sense amplifier, and data output driver. Except bit-cell array (IBA) and postdecoder (IPD), the rest four units are combined into a single block as a data output unit (IDO) due to their small sizes in height. Similarly, the tag subarray is composed of tag bit-cell array, postdecoder, and tag match unit. However, due to its small dimensional sizes, only about 9 percent of the data subarray in area, the tag subarray (ITA) is also treated as a single block in our thermal modeling. Between the subarrays, there are data/address routing units (IDA) and postdecode address routing units (IPA), which are used to route data and address. IVS stands for void space which does not have any power consumption. The floorplan of an instruction cache (at 70 nm technology) similar to the one in Alpha 21364 is given in Fig. 1a. The Alpha EV7 floorplan (with its EV6 core) from [31] is used as the reference floorplan for this study, which is shown in Fig. 1b.

5 Malicious Codes

To thermally attack the instruction cache, the generally known hotspots should remain cool below the DTM trigger temperature. Otherwise, DTM may be invoked by the thermal sensors near the traditional hotspots, which makes our attack failed. The basic philosophy of our malicious code is to largely disable the back end (e.g., register file and ALU that are known as the hottest functional units) of the processor datapath while keeping the instruction cache busy. Theoretically, the code that involves no actual operations in the processor's back end can be an effective malicious code for the above purpose. Therefore, the code which contains an NOP in an infinite loop could be the simplest malicious code for attacking the instruction cache.

However, if a malicious code consists of only one small basic block, it is not sufficient to attack the L1 instruction cache. If the microprocessor has an instruction buffer, it makes the processor access the instruction buffer instead of the L1 instruction cache, which also makes our malicious code unsuccessful. Thus, we should use at least more than two basic blocks which consist of NOP and BR instruction. Since each basic block resides in different cache lines, the L1 instruction cache should be accessed every cycle. To deal with large instruction buffers that can store multiple cache lines, we use N basic blocks. N should be sufficiently large considering the instruction buffer capacity. In this manner, we can make the instruction cache be accessed every cycle. To improve the efficiency of the malicious code, we use dummy codes between the basic blocks. Dummy codes are not executed because a program flow cannot reach them. Using the dummy codes, we can make basic blocks reside in the same IBA, which helps only specific data subarrays and tag subarrays to be accessed. Maximum N value is determined by the size of one IBA, since too many basic blocks cannot be stored all in one IBA. The range of N value is denoted as (1). We ignored the program initialization code section.

$$\frac{instruction\ buffer\ capacity}{cache\ line\ size} < N \leq \frac{one\ IBA\ capacity}{cache\ line\ size}. \quad (1)$$

(a)

(b)

Fig. 1. The floorplan of the instruction cache (unit size: cm) and the entire microprocessor. The cache configuration is from Alpha 21364. (a) Modeled floorplan of the instruction cache. (b) Modeled floorplan of Alpha 21364.

Fig. 2 shows the malicious code example. Though 512 was the maximum N value that we could use, we used 256 as N value because program initialization code section occupies some portion of our target IBA (IBA0). However, 256 is a sufficiently large value to make the instruction buffer useless. There is one big loop body which contains 256 basic blocks. Between the basic blocks, it contains dummy codes. We make the basic blocks reside in IBA0 in this code example. Since the access to IBA0 is directly related to the access to tag subarray0 (ITA0) and 1 (ITA1) among four ITAs, this code address alignment is also effective in heating a specific tag subarray as well as a specific data subarray. Please note that we assume that none of code optimization is applied in this code.

Though Alpha architecture automatically converts NOP instruction to BIS instruction, we used NOP instruction which does nothing in microprocessor's back end. Currently, microprocessor architects may make NOP do nothing in processor's back end to reduce power consumption and complexity.

```
Address                        Assembly code
                               .globl      main
                               .ent        main
                     main:     .context    full
                               ldq         $28, -16384($sp)
                               ldq         $28, -8256($sp)
                               lda         $sp, -16384($sp)
                               .frame      $sp, 16384, $26
                               .prologue   0
0x20003100           L$0:      nop
0x20003104                     br          L$1
   :                           (dummy codes)
0x20003140           L$1:      nop
0x20003144                     br          L$2
   :                           (dummy codes)
0x20003180           L$2:      nop
0x20003184                     br          L$3
   :                           (dummy codes)
   :                                 :
   :                                 :
   :                                 :
0x200070C0           L$255:    nop
0x200070C4                     br          L$0
   :                           (dummy codes)
                               .end        main
```

Fig. 2. An example of malicious code.

6 Simple Techniques to Protect Caches from Thermal Attacks

To detect thermal emergency, the simplest way is to deploy accurate thermal sensors as many as possible. Though we can protect the chip from thermal runaway using lots of thermal sensors, the cost overhead is too high to bear. Instead, we propose two cost-effective techniques. One is a hardware-based protection technique and the other is a software screening technique which leverages specific softwares such as the malicious code scanner [24].

6.1 Hardware-Based Protection Technique

One data subarray or tag subarray being continuously accessed can become a hotspot in the microprocessor. To avoid this fine-grain localized hotspot, we propose a simple and effective hardware-based solution. If the proposed instruction cache controller senses the fine-grain localized hotspot, an appropriate DTM can be invoked. Since there is a high correlation between access frequency and temperature, the access counter for each ITA and IBA in the cache controller can be used as a virtual thermal sensor. Compared to deploying many thermal sensors, this solution is much more cost-effective. As depicted in Fig. 3, when an access counter of a specific block (block access counter) exceeds the predefined threshold value in a predefined time slice (T_{slice}), the heavy access counter of this block is increased. In a subsequent time slice, if an access counter of this block exceeds the threshold again, the heavy access counter of this block is increased again. Otherwise, the heavy access counter of this block is reset to zero regardless of the current heavy access counter value. All of the block access counters are set to zero at each time slice boundary. If the heavy access counter of any block reaches predefined threshold value (H_{th}), the microprocessor stops fetching instructions until the microprocessor becomes sufficiently cool. In other words, if the microprocessor consecutively as well as intensively accesses the same data subarray or tag subarray, the microprocessor stops fetching for a sufficient cooling time. The relationship among the sufficient cooling time (T_c), the heavy access threshold (H_{th}), and the time slice (T_{slice}) is denoted as follows:

$$T_c = \alpha \times H_{th} \times T_{slice}. \quad (2)$$

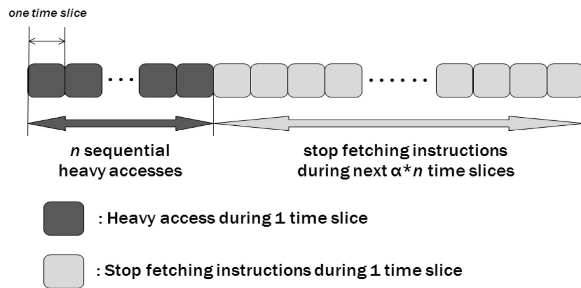

Fig. 3. The proposed hardware-based protection technique.

The value of α in (2) represents the aggressiveness of the protection technique. It depends on the amount of time needed to cool down the microprocessor. After the cooling time, all of the heavy access counters are set to zero, and then, the microprocessor resumes the execution. Since in the normal applications, the same tag subarray or data subarray is not likely accessed intensively, the proposed technique has little performance overhead. Additional hardware overhead is only small control logic for fetch throttling and two access counters (block and heavy) for each IBA and ITA. In this paper, 90,000 is used as the threshold value of the block access counter per time slice and heavy access threshold value (H_{th}) is set to 5. One time slice (T_{slice}) is set to 100,000 cycles and α is set to 2. Proper values for these parameters can be determined by thermal simulations [5].

6.2 Software Screening

Since the malicious codes introduced in Section 5 have specific patterns, the software such as malicious code scanner [24] can detect thermally malicious codes. Compared to the hardware-based protection techniques, the software screening technique can detect the malicious code before the malicious code is executed while hardware-based technique can detect thermal emergency after the malicious code is once executed. To prevent malicious codes from being executed, software techniques such as API hooking [15] can be adopted. By adding thermally malicious code signature [24] to currently used malicious code scanners, the software screening technique eliminates thermal threat in a cost-effective manner. However, since there are too many variations of the malicious codes, software screening should know all the possible patterns.

7 Evaluation

7.1 Evaluation Methodology

For experimental evaluation, we extended the original SimpleScalar simulator [4] to model the Alpha 21364 microprocessor as close as possible. The detailed configuration of the simulated Alpha 21364 microprocessor is given in Table 2. The power model for this Alpha 21364 microprocessor simulator is derived from Wattch [2]. HotSpot [31], [32] and HotLeakage [35] are also incorporated into this processor simulator to profile the temperatures and the leakage power of the caches. Please note that HotSpot has been validated against Floworks, which is a commercial simulator of 3D fluid and heat flow [32]. As mentioned in Section 4, the reference floorplan is from Alpha 21364. For the 70 nm technology we are evaluating here, the floorplan is shown in Fig. 1b and the dimensions of the processor components are scaled down from the 180 nm technology. We employed several parameters such as the supply voltage and frequency from IBM Power6 microprocessor [7], [9]. Other HotSpot-related parameters are from HotSpot 4.0 default parameters [13]. Several DTM-related parameters are same as we mentioned in Section 3.2 (DTM trigger and emergency temperature is 95°C and 100°C, respectively.). Please note that our simulated microprocessor is not a commercial microprocessor but a hypothetical microprocessor like Alpha EV7 with the advanced process technology.

TABLE 2
Parameters for the Simulated Alpha 21364 Processor

Processor Core	
Int/FP Issue Queue	20/15 entries
Load/Store Queue	64 entries
Int/FP Physical RegFile	80/72 registers
Fetch/Decode/Commit	4 instructions per cycle
Issue Width	6 instructions per cycle (4 Int+2 FP)
Function Units	4 IALU, 2 IMULT/IDIV, 2 FALU, 1 FMULT/FDIV/FSQRT, 2 Mem ports
Branch Predictor	
Branch Predictor	Tournament predictor, PAg/GAg with GAg chooser
BTB/RAS	2048 entries, 2-way / 32-entry
Memory Hierarchy	
L1 ICache/DCache	64KB, 2 ways, 64B blocks, 2 cycles
L2 UCache	4MB, 8 ways, 128B blocks,12 cycles
Memory	225 cycles first chunk, 12 cycles rest
I/DTLB	128 entries, full assoc., 30 cycle miss penalty
Power/Chip/Thermal Parameters	
Technology	70nm
Clock frequency	5.6GHz
Core Supply voltage	1.2V
SRAM Supply voltage	1.3V
Die thickness	0.15mm
Initial temperature	60°C

7.2 Thermal Attack by Malicious Codes

Fig. 4 depicts the thermal map (of peak temperature) of the simulated microprocessor when executing the malicious code shown in Fig. 2. The upper part of the instruction cache becomes thermal hotspot, while the other parts of the microprocessor become relatively cool. Fig. 5 shows the peak temperature of major

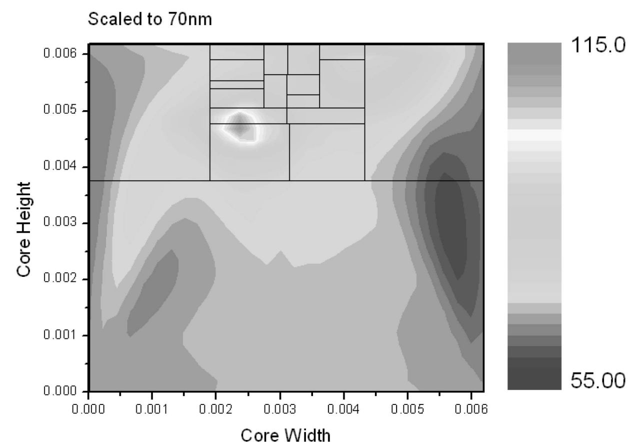

Fig. 4. Thermal map of the simulated microprocessor when executing the malicious code.

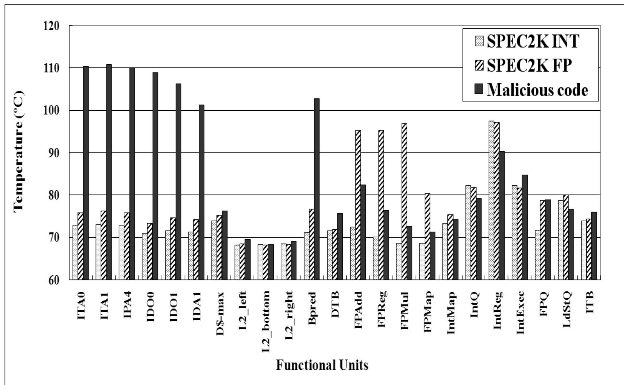

Fig. 5. The peak temperature of the six hottest units in the instruction cache and major hot functional units.

hot functional units in the microprocessor when executing the malicious code and the SPEC2000 benchmarks. As illustrated in Fig. 5, the temperature of ITA0, ITA1, and IPA4 which are located in upper part of the instruction cache is 110.31°C, 110.78°C, and 110.10°C, respectively. These units become the hottest units in the microprocessor. Besides the instruction cache, the branch predictor (Bpred) is the hottest unit which is 102.66°C. Among the nine units where the thermal sensors are deployed (these units are emphasized by gray-shaded, bold, and italic font in Table 1), the temperatures of these units are always below the DTM trigger temperature. It makes thermal sensors deployed in the generally known hotspots useless.

Comparing the temperature difference of the instruction cache between two cases (when executing the malicious code and the normal application), the maximum temperature difference is 37.85°C (ITA1). It implies that such a malicious behavior can abnormally overheat the instruction cache, which can incur malfunction or reliability problems [31] in the instruction cache. Additionally, since the leakage power is strongly related to the temperature [17], the leakage power also gets increasing with the malicious code.

The required time for heating the instruction cache (ITA1) to 100°C is 38.24 ms in 5.6 GHz frequency. Because recent operating systems such as UNIX or Linux use 50-200 ms as the context switch time slice [28], the malicious code can attack without being intervened by the context switch. Note that 100°C is the maximum junction temperature for 70 nm technology, as reported in ITRS 2006 [29].

7.3 Protecting the Instruction Cache from Thermal Attacks

The thermal map of the simulated microprocessor when adopting the protection technique described in Section 6.1 is depicted in Fig. 6. After adopting the proposed technique, the temperature difference between the upper part of the instruction cache and the other functional units becomes very small. When using the proposed technique, the temperature of ITA1, ITA0, and IPA4, which were the localized hotspots, is reduced to 84.5°C, 84.35°C, and 83.95°C, respectively, as shown in Fig. 7. The average temperature drop of the seven units in Fig. 7 is 26.13°C. After adopting this simple protection technique, the instruction cache becomes free of potential thermal risk.

As discussed in Section 6.2, software screening technique can detect thermally malicious codes thus preventing them from execution, consequently protecting the instruction cache from thermal attacks. The detailed evaluation of the software screening technique is out of the scope of this paper.

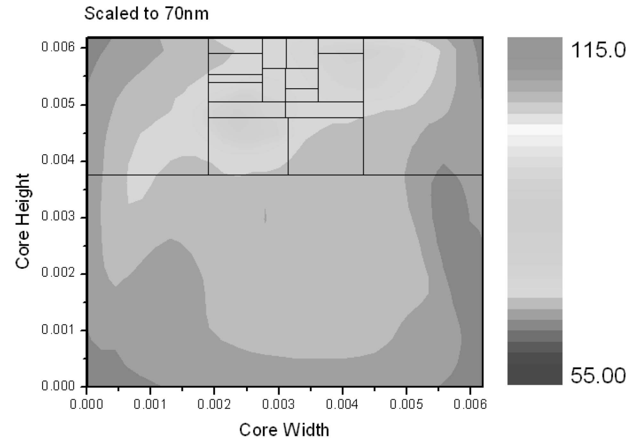

Fig. 6. Thermal map of the simulated microprocessor when protected by the proposed protection technique.

8 Conclusions and Future Works

In this paper, we demonstrated how to thermally attack the microprocessor. We showed that it can incur a serious thermal reliability problem in microprocessors. Furthermore, we also evaluated the effectiveness of the proposed schemes that protect the instruction cache against potential thermal attacks. Note that the proposed techniques have little performance overhead, since there are very few cases such as the malicious behavior in the normal applications. A major focus of this work is to identify the design deficiency of thermal security in the conventional thermal management schemes, which is vulnerable to potential thermal attacks. It is also fundamentally different from previous efforts on DTM, whose goal is to prevent the well-known hottest processor components from incurring the possible thermal emergency situation.

As technology scales down and power density grows up, thermal emergency by malicious codes will be more severe. Therefore, computer security engineers as well as microprocessor designers should be aware of potential thermal attack caused by thermal malicious codes, which may lead microprocessors to thermal runaway. The malicious code scanner should be able to detect thermally malicious codes given the availability of patterns of malicious codes. Though a simple pattern of malicious codes targeting at the instruction cache is shown in this paper, various patterns of malicious codes can be generated by an automatic code generation tool that will be implemented by our research group soon.

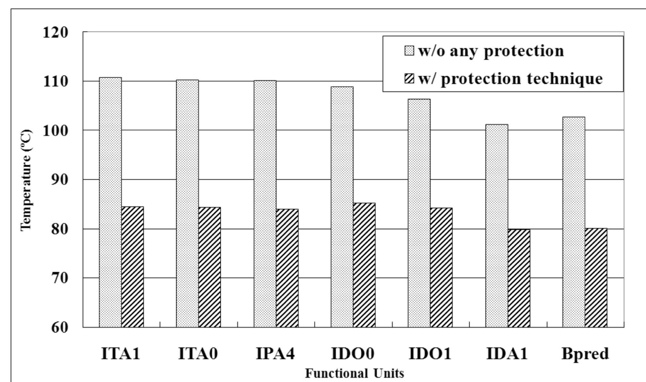

Fig. 7. Comparison of peak temperatures.

ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. R01-2007-000-20750-0). This work was also supported by a Korea University Grant. The authors would like to thank Professor Kevin Skadron for his helpful comments. Finally, they would like to thank the anonymous reviewers for their helpful feedback. Sung Woo Chung is the corresponding author of this paper.

REFERENCES

[1] S. Borkar, "Design Challenges of Technology Scaling," *IEEE Micro,* vol. 19, no. 4, pp. 23-29, July/Aug. 1999.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural Level Power Analysis and Optimizations," *Proc. Int'l Symp. High-Performance Computer Architecture,* 2000.

[3] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," *Proc. Int'l Symp. High-Performance Computer Architecture,* Jan. 2001.

[4] D. Burger, A. Kagi, and M.S. Hrishikesh, "Memory Hierarchy Extensions to Simplescalar 3.0," Technical Report TR99-25, Dept. of Computer Sciences, Univ. of Texas at Austin, 2000.

[5] S.W. Chung and K. Skadron, "Using On-Chip Event Counters for High-Resolution, Real-Time Temperature Measure," *Proc. IEEE/Am. Soc. Mechanical Engineers (ASME) 10th Intersoc. Conf. Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm '06),* May 2006.

[6] P. Dadvar and K. Skadron, "Potential Thermal Security Risks," *Proc. IEEE Semiconductor Thermal Measurement, Modeling, and Management Symp. (Semi-Therm 21),* pp. 229-234, Mar. 2005.

[7] J. Davis, D. Plass, P. Bunce, Y. Chan, A. Pelella, R. Joshi, A. Chen, W. Huott, T. Knips, P. Patel, K. Lo, and E. Fluhr, "A 5.6 GHz 64 kB Dual-Read Data Cache for the POWER6™ Processor," *Proc. Int'l Solid-State Circuits Conf. (ISSCC), Digest of Technical Papers,* Feb. 2006.

[8] M.S. Floyd, S. Ghiasi, T.W. Keller, K. Rajamani, F.L. Rawson, J.C. Rubio, and M.S. Ware, "System Power Management Support in the IBM POWER6 Microprocessor," *IBM J. Research and Development,* vol. 51, no. 6, pp. 733-746, 2007.

[9] J. Friedrich et al., "Design of the POWER6 Microprocessor," *Proc. Int'l Solid-State Circuits Conf. (ISSCC), Digest of Technical Papers,* Feb. 2007.

[10] S.H. Gunther, F. Binns, D.M. Carmean, and J.C. Hall, "Managing the Impact of Increasing Microprocessor Power Consumption," *Intel Technology J.,* vol. 5, no. 1, p. 9, Feb. 2001.

[11] J. Hasan, A. Jalote, T.N. Vijaykumar, and C.E. Brodley, "Heat Stroke: Power-Density-Based Denial of Service in SMT," *Proc. Int'l Symp. High-Performance Computer Architecture,* 2005.

[12] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas, "A Framework for Dynamic Energy Efficiency and Temperature Management," *Proc. Int'l Symp. Microarchitecture,* 2000.

[13] W. Huang, K. Sankaranarayanan, K. Skadron, R.J. Ribando, and M.R. Stan, "Accurate Pre-RTL Temperature-Aware Design Using a Parameterized, Geometric Thermal Model," *IEEE Trans. Computers,* vol. 57, no. 9, pp. 1277-1288, Sept. 2008.

[14] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," *Proc. IEEE/ACM Int'l Symp. Microarchitecture,* Dec. 2003.

[15] I. Ivanov, *API Hooking Revealed,* http://www.codeproject.com/system/hooksys.asp, 2002.

[16] J.K. John, J.S. Hu, and S.G. Ziavras, "Optimizing the Thermal Behavior of Subarrayed Data Caches," *Proc. IEEE Int'l Conf. Computer Design,* Oct. 2005.

[17] N.S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan, "Leakage Current: Moore's Law Meets Static Power," *Computer,* vol. 36, no. 12, pp. 68-75, Dec. 2003.

[18] J.C. Ku, S. Ozdemir, G. Memik, and Y. Ismail, "Thermal Management of On-Chip Caches through Power Density Minimization," *Proc. IEEE/ACM Int'l Symp. Microarchitecture,* Nov. 2005.

[19] K.-J. Lee, K. Skadron, and W. Huang, "Analytical Model for Sensor Placement on Microprocessors," *Proc. IEEE Int'l Conf. Computer Design,* Oct. 2005.

[20] K.-J. Lee and K. Skadron, "Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors," *Proc. Workshop High-Performance, Power-Aware Computing (HP-PAC),* Apr. 2005.

[21] A. Merkel, F. Bellosa, and A. Weissel, "Event-Driven Thermal Management in SMP Systems," *Proc. Second Workshop Temperature-Aware Computer Systems (TACS '05),* June 2005.

[22] R. Mukherjee and S.O. Memik, "Systematic Temperature Sensor Allocation and Placement for Microprocessors," *Proc. Design Automation Conf.,* July 2006.

[23] N. Paul, S. Gurumurthi, and D. Evans, "Thermal Attacks on Storage Systems," *Proc. 23rd IEEE Conf. Mass Storage Systems and Technologies (MSST '06),* May 2006.

[24] C. Pfleeger and S. Pfleeger, *Security in Computing,* third ed. Prentice-Hall PTR, 2003.

[25] 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 Language and Operating System (ASPLOS '04),* Oct. 2004.

[26] G. Reinman and N. Jouppi, "An Integrated Cache Timing and Power Model," technical report, Compaq Western Research Laboratory, 1999.

[27] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '02),* 2002.

[28] R.S. Shindi and S. Cooper, "Evaluate the Performance Changes of Processor Simulator Benchmarks When Context Switches are Incorporated," *Proc. ACM SIGAda '06,* Nov. 2006.

[29] *SIA,* Int'l Technology Roadmap for Semiconductors (ITRS), Available at http://www.itrs.net/reports.html, 2006.

[30] K. Skadron, T. Abdelzaher, and M.R. Stan, "Control-Theoretic and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," *Proc. Int'l Symp. High-Performance Computer Architecture,* Feb. 2002.

[31] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayana, and D. Tarjan, "Temperature-Aware Microarchitecture," *Proc. Int'l Symp. Computer Architecture (ISCA '03),* June 2003.

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

[33] J. Srinivasan and S.V. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications," *Proc. Int'l Conf. Supercomputing (ICS '03),* June 2003.

[34] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing Power in High-Performance Microprocessors," *Proc. Design Automation Conf.,* 1998.

[35] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotleakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects," technical report, Dept. of Computer Science, Univ. of Virginia, 2003.

[36] VAR Business, *Intel Clears Up Post-Tejas Confusion,* Available at http://www.varbusiness.com/sections/news/breakingnews.jhtml?articleId=18842588, 2009.