Thermal-Aware Memory Mapping in 3D Designs

Ang-Chih Hsieh and TingTing Hwang
Department of Computer Science, National Tsing Hua University
HsinChu, Taiwan 300

***Abstract*—DRAM is usually used as main memory for program execution. The thermal behavior of a memory block in a 3D SIP is affected not only by the power behavior but also the heat dissipating ability of that block. The power behavior of a block is related to the applications run on the system while the heat dissipating ability is determined by the number of tier and the position the block locates. Therefore, a thermal-aware memory allocator should consider the following two points. First, allocator should consider not only the power behavior of a memory block but also the physical location during memory mapping, second, the changing temperature of a physical block during execution of programs. In this paper, we will propose a memory mapping algorithm taking into consideration the above-mentioned two points. Our technique can be classified as static thermal management to be applied to embedded software designs. Experiments show that our method can reduce temperature of memory system by 17.2°C as compared to a straightforward mapping in the best case, and 13.4°C in average.**

I. Introduction

System in package (SIP) provides a cost-effective solution for large-scale integration [1]. This technology has been widely used in mobile devices and embedded systems. Current technology allows more than twenty chips to be stacked in one package [2]. With the capacity provided by SIP technology, integrating memory chips into package has become popular in recent years. Several researches on memory integration based on SIP have been studied [3]–[7]. Though SIP technology provides extremely high capacity for circuit integration, it suffers severe thermal stress because of three dimensional stacking of ICs [8]. Thermal stress will induce variation of DRAM retention time and reliability problem [9].

Many temperature-aware researches have been conducted. They can be classified into two categories, dynamic and static thermal managements. The former techniques detect the temperature information at run-time, and stop hot units operating till their temperature cools down. Examples such as voltage scaling [10], throttling techniques [11], and non-DVS localized thermal management [12] are in this category. Dynamic thermal management schemes can precisely monitor temperature value and guarantee that the system temperature will never be higher than a predefined constraint, however, at the cost of slowdown of the processor execution. As to static thermal management, the profiling data is generated first and then used to analyze the temperature distribution of the program. [13] proposes a floorplan technique from microarchitecture level point of view to reduce the hotspot temperature. This floorplan algorithm determines the locations of functional units by spreading hot functional units and surrounding them by cooler functional units. However, this technique is less flexible because the same floorplan is used for all applications and the locations of functional units can not be changed after floorplan is done. Yet, another approach [14] is proposed from compiler-level point of view which distributes computations to different functional units so that the hotspots are prevented.

Except [15], none of previous research addressed thermal and energy problem for 3D memory design. In [15], energy and delay savings due to 3D partition of cache memory based on wafer-bounding technology is discussed. Although its method is suitable for custom cache design, it cannot be applied to DRAM chips in stacked SIP design.

This work was supported in parts by Synopsys Inc. and National Science Council of Taiwan, Republic of China, under grant NSC 97-2220-E-007-033.

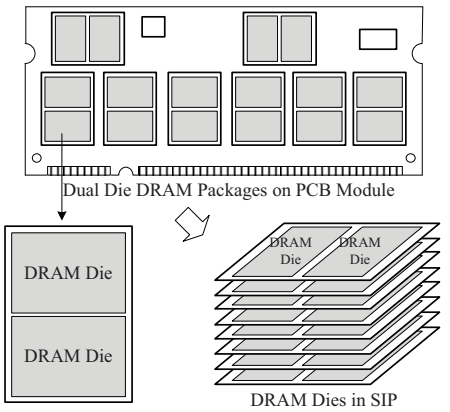


Fig. 1. DRAM Packages on PCB and DRAM Dies in SIP Design

DRAM is usually used as main memory for program execution. The thermal behavior of a memory block in a 3D SIP is affected not only by the power behavior but also the heat dissipating ability of that block. The power behavior of a block is related to the applications run on the system while the heat dissipating ability is determined by the number of tier and the position the block locates.

Therefore, a thermal-aware memory allocator should consider the following two points. First, allocator should consider not only the power behavior of a memory block but also the physical location during memory mapping, second, the changing temperature of a physical block during execution of programs. In this paper, we will propose a memory mapping algorithm taking into consideration the above-mentioned two points. Our technique can be classified as static thermal management to be applied to embedded software designs.

The rest of the paper is organized as follows. In Section II, motivation of this work is presented. Section III describes our system model and problem definition. In Section IV, details of each step of our algorithm are introduced. These techniques include thermal aware memory configuration, program behavior analysis and ILP formulation. The experimental results are given in Section V. Finally, Section VI concludes this work.

II. Observation & Motivation

In traditional on-board designs, DRAM chips are placed in a planar space. Therefore, system designers can view all DRAM chips identical and assume that all DRAM chips have the same heat dissipating ability. However, when DRAM chips are stacked using SIP technology, chips of different tiers have very different environmental conditions. For example, in 3D memory, it is more difficult to dissipate the power of a physical block on the middle tier than on the top tier, as shown in Figure 1. That is, chips on different tiers have different heat dissipating abilities and can sustain different access frequencies under a given temperature constraint. Moreover, for each access, multiple banks of different chips need to be triggered at the same time. How to select a bank in a chip should consider thermal issue.

978-3-9810801-5-5/DATE09 © 2009 EDAA

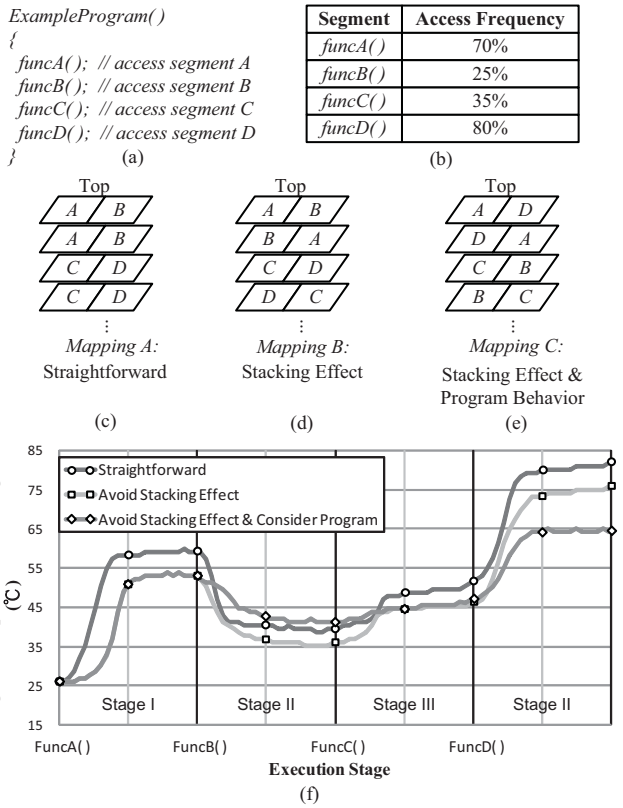


Fig. 2. (a) Example Program; (b) Access Frequency; (c) *Mapping A*; (d) *Mapping B*; (e) *Mapping C*; (f) Simulation Result

On the other hand, the logical memory space for an application comprises several memory blocks for data, instructions, heap and stack. Each block has different access behavior and access frequency. And even different segments in a block can have quite different access frequencies. For example, instructions of an application are all loaded to a consecutive memory space. But segments for instructions of different loops or different functions are accessed with different frequencies. This situation can also be found in memory blocks for data, heap and stack. In traditional on-board DRAM chips, the mapping between these memory blocks and physical DRAM chips can be simple since all DRAM chips are identical. However, for SIP designs, the mapping problem becomes complicated because the behavior of each memory block and the heat dissipating ability of each DRAM chip need to be considered simultaneously for thermal management.

Figure 2 gives an example to present our motivation. Assume that a program is executed with 4 stages. 4 functions named *funcA()*, *funcB()*, *funcC()* and *funcD()* are called in each stage, as shown in Figure 2(a). When a function is called, its corresponding memory segment is accessed. Since different function has different behavior, each segment has different access frequency. Let the access frequency of each segment be given in Figure 2(b) where access frequency is defined as the number of accesses to a memory segment divided by the total cycle counts of that stage. In this simplified example, we assume each memory die has only two banks. Due to design constraints, for each memory die, only one bank can be accessed at a time. Let a wider memory word be composed of bits from two dies. Then 2 memory dies are required to be triggered simultaneously for each access. This means an address will map to 2 banks of 2 different memory dies. Three mapping policies are shown in Figure 2(c)-(e). Figure 2(c) shows a straightforward mapping (*Mapping A*) where

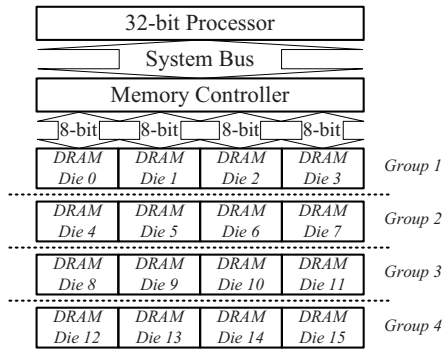


Fig. 3. Memory System

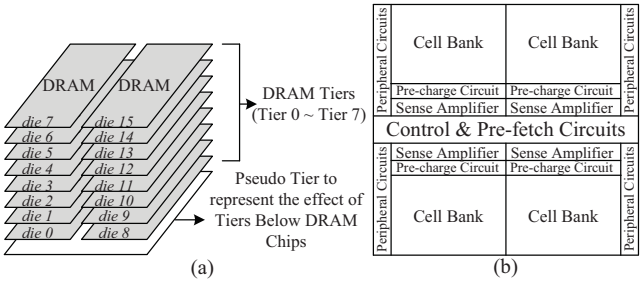


Fig. 4. (a) SIP Model; (b) Floorplan of a Typical DRAM chip

two banks at the same relative position denoted as A, B, C, D are accessed simultaneously. Figure 2(d) shows a mapping (*Mapping B*) to avoid stacking effect where banks accessed at the same tier are not in the same vertical position and Figure 2(e) a mapping (*Mapping C*) consider stacking effect and the access frequency where segments with high access frequency are mapped to banks on upper tiers. The simulation result by HotSpot 4.0 [16] is presented in Figure 2(f). The y-axis represents the highest temperature in all dies and the x-axis represents the execution stages referred as Stage I, II, III and IV. Each stage represents the execution period of each function. Stage I (*funcA()*) shows that the temperature is reduced at most 7°C by mappings considering stacking effect (*Mappings B, C*) as compared to *Mapping A*. Stages II & III (*funcB()* & *funcC()*) show that mappings considering stacking effect but banks located at bottom tiers (*Mapping C*) sometimes has higher temperature than straightforward mapping. But in both stages, the temperature is relative low because of low access frequency. The maximum temperature occurs in Stage IV because of the highest access frequency. Stage IV (*funcD()*) shows that a mapping considering stacking effect (*Mapping C*) and program behavior can reduce the maximum temperature by 18°C and 12°C as compared to *Mappings A* and *B* respectively.

III. System Model and Problem Definition

In this section, we will first give our system model. Based on the model, we will define our problem and propose an overall design flow. The data width of a modern DRAM chip often ranges between 2^0-bit to 2^4-bit while processors have a 32-bit, 64-bit, or more data lines. Therefore, to read or write a 32-bit, 64-bit or more bit word from memory, multiple DRAM chips need to be accessed. Figure 3 gives an example of a system containing 32-bit processor, system bus, memory controller and 8-bit DRAM chips. To access a 32-bit data, 4 DRAM chips need to be activated simultaneously. Let the DRAM chips activated simultaneously form a *group*. Then, in the example, *DRAM Die 0* to *DRAM Die 3* are in the same *group*. To increase the number of words (address space) in the system, multiple *groups* are assembled. In the example, there are 4 *groups*. Hence, the total address space is 4 times the word capacity of one *group*.

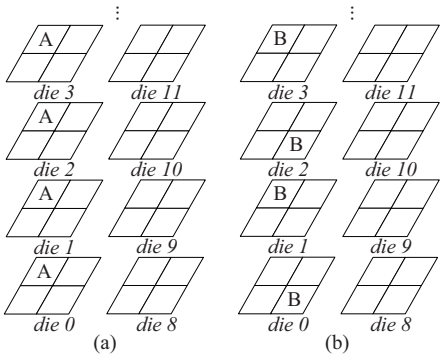


Fig. 5. Example for Memory Access

In a stacked SIP system, memory dies are stacked one tier on another. In one tier, there will be one or more dies packed. Due to intra-tier package routing constraint, the number of dies packed in one tier is rarely greater than 4. Figure 4(a) shows a system that has 8 tiers and 2 dies packed in one tier. Within a die, there are multiple banks in it. The floorplan of a typical DRAM chip with 4 banks is shown in Figure 4(b). For each access to a memory chip, *Control & Pre-fetch Circuits* block is always triggered. This block contains control, error correction and pre-fetch circuits. The *Cell Bank* block, the *Peripheral Circuits* block and the *Sense Amplifier* block of each bank will be triggered if that bank is accessed. In each DRAM die, **only one bank can be accessed at a time** due to the shared hardware and bus lines.

For a given address, memory controller will generate appropriate control signals to first select dies (forming a *group*) and then within dies to select banks. Let us take Figure 5 as an example using the same system configuration as shown in Figure 3 where 4 dies form a *group*. In Figure 5(a), *die 0*, *die 1*, *die 2* and *die 3* form a *group*. The banks in a *group* that are selected simultaneously to form a wider word are denoted as a *set*. Figure 5(a) shows that banks denoted as A in the same relative position form a *set*. In Figure 5(a), 4 dies form a *group* and there are 4 *sets* in a *group*.

However, in a stacked SIP design, Figure 5(a) will suffer serious thermal problem. The reason is as follows. Of all blocks in a die, *Sense Amplifier* block has extremely high power density due to their small area size. In general, more than 30% power of a DRAM chip is consumed by *Sense Amplifier* block while the area of a block is usually less than 5% of the total area. *Sense Amplifier* blocks are usually candidates for hotspot. If continuous addresses in a bank are accessed, *Sense Amplifier* blocks stacked at the same relative position in 3D space will result in high temperature.

On the other hand, Figure 5(b) shows another access mapping where the same dies form a *group* but banks in different relative positions are selected to form a *set*. In this mapping, lower temperature can be expected because the activated banks are not in the same vertical location.

In this paper, we will study a memory mapping problem to minimize the maximum temperature in a stacked 3D memory system. The problem is defined as follows. Given parameters of a memory system and the profiling of memory references for all application programs, the objective is to find a memory configuration and a mapping from logical address to physical location so that the maximum temperature is minimized.

To solve this problem, the flow depicted in Figure 6 is proposed. The first step, *Determination of Candidate Configurations* is, for given parameters of a memory system, to find candidate memory configurations (in Section IV-A). Then, behaviors of applications run on the system are analyzed in the second step, *Application Behavior Analysis*, where logical memory blocks that have the similar behaviors are grouped in a *segment* (in Section IV-B). According to the candidate configurations and *segments* obtained, the last step, *ILP Formation for Segments Mapping*, applies ILP techniques to perform mapping so that the maximum temperature is minimized (in Section IV-C).

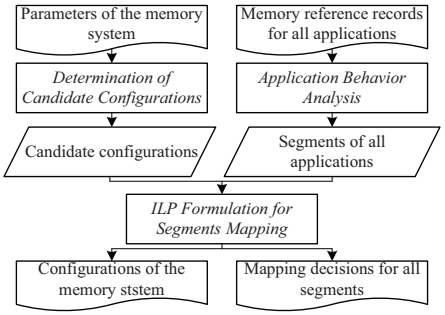


Fig. 6. Overall Flow

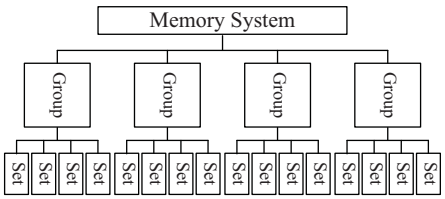


Fig. 7. Hierarchical View

IV. Thermal Driven Memory Address Mapping Algorithm

Before we present our mapping algorithm, we first review some terms defined in Section III.

group for a given address, the dies that are accessed simultaneously form a *group*.

set for a given address and a given *group*, the banks that are accessed simultaneously form a *set*.

segment a collection of consecutive logical memory blocks that have similar behaviors is called a *segment*.

The parameters of a memory system include the number of tiers, $\#tier$, the number of DRAM dies on one tier, $\#die\_on\_tier$, the number of banks in a DRAM die, $\#bank$, the bit width of a DRAM die, $\#bit\_width\_die$, the size of a DRAM die, $\#bit\_die$, and the bit width of system bus, $\#bit\_width\_system$. $\#bit\_width\_system/\#bit\_width\_die$ determines the number of DRAM dies in a $group$ and also the number of banks in a set. The number of words in a set is computed as $\#bit\_die/(\#bit\_width\_die \times \#bank)$.

A. Determination of Candidate Configurations

For given parameters of a memory system, we need to determine how to form a *group* and how to form a *set* within a *group*. Let us take the system in Figures 3 and 4 as an example. In this example, because of $\#bit\_width\_system = 32$ and $\#bit\_width\_die = 8$, the number of dies in a *group* is 4. There are 4 banks in a die. Hence, the number of *sets* in a *group* is 4. Figure 7 gives the hierarchical view of the system.

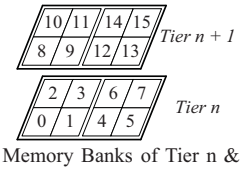


Fig. 8. Memory Banks of Tier n & Tier n + 1

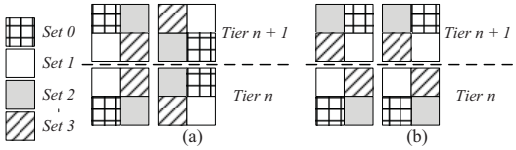


Fig. 9. (a) *Configuration I*; (b) *Configuration II*

First, we show how to form a *group*. Intuitively, we can select any 4 dies to form a *group*. However, most of combinations of dies are not required to be considered. Because dies in a *group* are accessed simultaneously, thermal behavior of a *group* is determined by the die that has the worst behavior. For example, if *die 7*, *die 6*, *die 5* and *die 4* in Figure 4(a) are defined as a *group*, though *die 7* is on the top tier and has the best heat dissipating ability, the actual thermal behavior of the *group* is bounded by *die 4*. No matter how low the temperature of *die 7* is, the memory space provided by the *group* would not be functional if *die 4* is overheated. Thus, dies in a *group* should have similar environmental conditions.

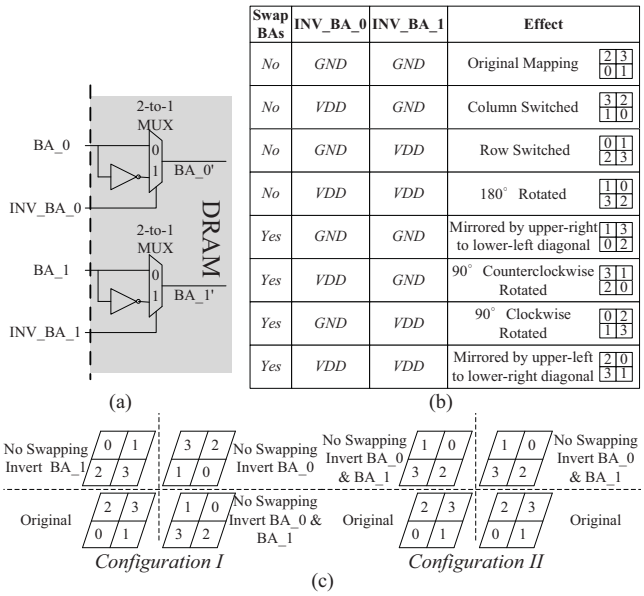
Based on the discussion above, how to form a *group* becomes straightforward. We should group dies on consecutive tiers into a *group*. In our example, because the number of dies in a *group* = 4 and $\#die\_on\_tier = 2$, dies on 2 neighboring tiers forms a *group*. That is, *die 0*, *die 1*, *die 8*, and *die 9* form a *group*, and *die 2*, *die 3*, *die 10*, and *die 11* form a *group*,...etc.

Next, we show how to determine the banks in a *set*. First, banks on the same tier have different heat dissipating abilities when $\#die\_on\_tier \geq 2$. For example, suppose there are two dies on a tier as shown in Figure 8. Banks 1, 3, 4 and 6 are in the middle area of the tier and therefore have worse thermal behavior than banks 0, 2, 5 and 7. Second, accessing banks of different dies at the same vertical position will result in undesirable thermal effect. For example, banks 0, 8 are at the same vertical position. If they are accessed simultaneously, heat will be generated in a small area and cannot be dissipated in vertical directions. This situation should be avoided. Based on the discussion above, possible *sets* combinations for a *group* can be defined through enumeration. The term *configuration* is used to refer to a definition of all *sets* in a *group*. We use the example in Figure 8 to explain how to determine possible *configurations* where dies on two neighboring tiers form a *group*.

We start with defining a *set* with best thermal behavior. As mentioned earlier, the thermal behavior of a *set* is determined by the bank with the worst thermal behavior. Therefore, to define a *set* with best thermal behavior, two rules should be followed. Rule 1 is that banks in the middle area should not be grouped in the same *set* and rule 2 is that banks in the same vertical position should not be grouped in the same *set*. Following these two rules, Figure 9 shows two resultant *configurations*, *Configuration I* and *Configuration II*, where banks drawn in the same patterns are defined as a *set*. Two *configurations* have their own characteristics. In *Configuration I*, *set 0* and *set 1* have good heat dissipating ability because the banks in these two *sets* are all in the boundary. However, the environmental conditions of *set 2* and *set 3* are worse than those of *set 0* and *set 1* because banks in *set 2* and *set 3* are all located in the middle positions with less heat dissipating abilities. On the other hand, in *Configuration II*, the thermal behavior of each *set* is almost identical.

Configuration I is suitable for a program with uneven access to memory while *Configuration II* is good for a program with even memory access. Which *configuration* to choose will depend on the behavior of the programs executing on the system. Thus, both *configurations* are selected as candidate *configurations* in our example.

Nevertheless, the *configurations* we obtained do not comply with the design of an off-chip DRAM design. In traditional designs, a



Swap BAs	INV_BA_0	INV_BA_1	Effect
No	*GND*	*GND*	Original Mapping
No	*VDD*	*GND*	Column Switched
No	*GND*	*VDD*	Row Switched
No	*VDD*	*VDD*	180° Rotated
Yes	*GND*	*GND*	Mirrored by upper-right to lower-left diagonal
Yes	*VDD*	*GND*	90° Counterclockwise Rotated
Yes	*GND*	*VDD*	90° Clockwise Rotated
Yes	*VDD*	*VDD*	Mirrored by upper-left to lower-right diagonal

Fig. 10. (a) Re-mapping Logic; (b) Re-mapping Table; (c) Example

common address bus is used for all memory dies. Therefore, bank address of each memory die is identical and banks at the same vertical locations are always accessed as a *set*. To have different banks in different dies accessed simultaneously, different bank addresses are required for each die. It is not feasible for memory controller to generate different bank addresses for each die due to in-package routing overhead. A re-mapping circuit shown in Figure 10(a) is proposed to be added to each DRAM die. Let *BA_0* and *BA_1* stand for the input pads for bank address *bits 0* and *1*. By setting *INV_BA_0* and *INV_BA_1* to *VDD* or *GND*, we can select to invert bank address *bits 0* and *1* or not. The re-mapped bank address bits are denoted as BA_0' and BA_1' which are sent to control circuit and determines the bank to be accessed. Moreover, swapping address lines connected to *BA_0* and *BA_1* doubles the mapping space. Table in Figure 10(b) enumerates all mappings supported by the proposed circuit. The first column of the table specifies whether the address lines are swapped, and the second and third columns represent whether *INV_BA_0* and *INV_BA_1* are set to 1 or 0. The forth column gives the address of each bank after re-mapping. Though only one thirds of all possible mappings are supported by our proposed circuit, it is sufficient to implement most of desired *configurations*. Figure 10(c) shows the settings for *Configurations I* & *II* as examples.

Next, we should determine the cost of each *configuration* under different access frequency to each *set*. In each *configuration* and in each *set*, we define the relation between temperature and access frequency by simulation. This relation can be used to determine the cost of mapping a memory *segment* with given access frequency to a *set*. For a *set*, the average power is defined as follows. First, the access to memory is divided to read access and write access. And operating power in Equation (1) considers different ratios of read and write access where α represents the ratio of read access to total access and $(1-\alpha)$ the ratio of write access to total access.

$$Power_{Operating} = Power_{Read} \times \alpha + Power_{Write} \times (1-\alpha) \quad (1)$$

Next, with different access frequency to a set, f, Equation (2) is defined for the average power.

$$Power_{Avg} = Power_{Operating} \times f + Power_{Standby} \times (1-f) \quad (2)$$

Finally, the simulation of each *set* is done as follows. For each f, the average power is calculated. Then, the hardware blocks of the

target *set* for simulation are set with the average power while all other blocks with standby power. Next, thermal simulation tool is called to obtain the steady state temperature. In this paper, HotSpot 4.0 [16] is used as our thermal simulation tool. The temperature obtained will be used to evaluate the effect of mapping a memory *segment* with access frequency f to a *set*. Notice that this temperature computed may be underestimated since all other surrounding blocks are assumed to be idle. That means, the interaction effect of blocks in the model is ignored. However, this underestimation is acceptable since the temperature can still reflect the thermal behavior of a *set* under a given access frequency. We use the term,

$$T(j, f)$$

to represent the steady state temperature when *set* j is accessed with frequency f.

B. Application Behavior Analysis

For each program runs on the system, the memory requirement is varying over the time. We can partition a program's logical address space to a number of *segments* each with different access frequencies and then based on access frequency, map each *segment* to different physical locations in a 3D memory to minimize maximum temperature.

An algorithm, *Behavior Analysis Algorithm*, is developed for this purpose as shown in Figure 11. First, profiling of memory references for application programs is recorded. For each cycle, whether memory is accessed and if yes, which memory address is referenced are recorded. Next, the memory reference profiling is fed to our algorithm for analysis.

In each cycle, the algorithm first checks whether it has a memory reference. If yes, it then checks if there exists a *segment* containing the address of the reference (line 11). If no such *segment* exists, a new *segment* is created for the reference (lines 12, 13). If there does exist a *segment* containing the address of the reference, then update the access information to that *segment* (line 15). Since a *segment* may have different behaviors for different periods of time, *segments* need to be analyzed periodically. A variable named *counter* is presented to invoke *segments* analysis and *segments* merging for a fixed period of time (lines 17, 18, 19, 20, 21, 22). $Counter$ is a user defined variable and is increased by 1 every execution cycle. When $counter$ equals to $period$ where $period$ is a constant, *segments* analysis is invoked and $counter$ is reset to 0. By the time, the access frequency of each *segment* is computed as the number of memory accesses over the cycle counts.

Segments merging is used to merge neighboring *segments* which have similar behaviors to form a larger *segment*. Here, the criterion for merging is changing over the time. At the beginning of the algorithm, segments can be merged only when they are referencing adjacent address space and the access frequencies are identical. When in the later stages of the algorithm, the criterion for merging is looser. A threshold of access frequency is defined. As long as the difference of access frequencies is smaller than the threshold, two *segments* are merged. When the program completes, a number of *segments* with their access frequencies over different periods are obtained. For each *segment*, the highest frequency of all periods is then determined as the frequency of the *segment*.

C. ILP Formulation for Segment Mapping and Group Configuration

After we found candidate *configurations* for each *group* and analyzed the behaviors of programs, how to select a most suitable *configuration* for each *group* and how to map each memory *segment* to an appropriate *set* remain to be solved. Since this is an assignment problem and all constraints are linear, we can use an ILP formulation

```
1  Algorithm : Program Behavior Analysis Algorithm()
2  Input : Memory reference record
3  Output : Memory segments
4
5  counter = 0;
6  While(end of record is not reached)
7  {
8      ref = ReadNextReference();
9      If(ref is TRUE)
10     {
11         segment = FindSegmentFor(ref);
12         If(segment == NULL)
13             CreateNewSegmentFor(ref);
14         Else
15             AddInfoTo(segment, ref);
16     }
17     counter++;
18     If(counter == period)
19     {
20         counter = 0;
21         UpdateAllSegments();
22         MergeNeighboringSegmentsWithSimilarBehavior();
23     }
24 }
25 UpdateAllSegments();
26 MergeNeighboringSegmentsWithSimilarBehavior();
```

Fig. 11. Algorithm for *Behavior Analysis Algorithm*

to solve this problem if the objective function is also linear. By defining the cost of mapping a *segment* to a *set* based on the temperature $T(j, f)$ defined in Section IV-A, the objective function to be minimized can be defined as the summation of all mapping costs. The problem can then be solved by an ILP solver.

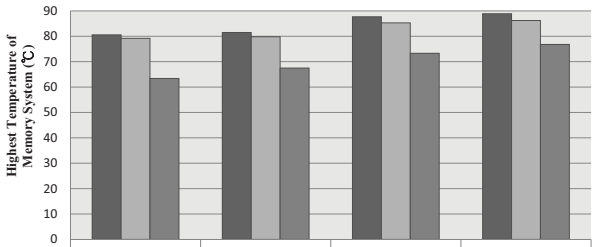
V. Experimental Results

In this section, experimental results for different execution conditions are presented. The system parameters are listed in Table I. We assume the system supports multiprogramming with Round-Robin scheduling and all programs run on the system are pre-loaded to memory. The program set is composed of MediaBench [17], PowerStone [18] benchmark suites and JM H.264/AVC CODEC [19]. The programs are duplicated to multiple instances to simulate systems with different memory utilization ratio. SimpleScalar 3.0 [20] is used to generate memory reference records. lp_solve 5.5 [21] is used as our ILP solver. HotSpot 4.0 [16] is used as our thermal simulation tool. To demonstrate the efficiency of our method, two straightforward mappings are tested for comparison. The first one selects 4 DRAM dies at the same relative positions of 4 consecutive tiers as a *group* and the second selects all 4 dies of 2 consecutive tiers as a *group*. Notice that no additional re-mapping circuits are added in these two mappings and therefore stacking effect among banks cannot be avoided. These two mappings are referred as M_1 and M_2 while our proposed mapping is referred as M_ours in the following discussion.

The first two experiments are used to observe the efficiency of our method under different memory utilization ratios. In these two experiments, the frequency of the processor is set to 800 MHz, which is 4 times the frequency of DRAM dies. As shown in the left most column of Figure 12, when memory utilization ratio is 75%, M_ours has the temperature reduction by 17.2°C and 15.8°C as compared to M_1 and M_2. This improvement is due to 25% unused memory space which provides more mapping flexibility. On the other hand, M_1 and M_2 not only suffer the stacking effect of banks but also have locations with less heat dissipating abilities. For example, M_1 maps a *segment* with high access frequency to 4 banks in the middle

TABLE I
SYSTEM PARAMETERS FOR EXPERIMENTS

System Parameter	Value
$\#tier$	8
$\#die\_on\_tier$	2
$\#bank$	4
$\#bit\_width\_die$	8
$\#bit\_width\_system$	32
DRAM Parameter	**Value**
Capacity	512 Mb
Clock Rate	200 MHz
Total Memory Size	**1 GB**



	75%, 800 MHz	95%, 800 MHz	75%, 1.2 GHz	95%, 1.2 GHz
M_1	80.6	81.5	87.7	88.9
M_2	79.2	79.8	85.3	86.2
M_ours	63.4	67.5	73.3	76.8

Fig. 12. Comparisons of the Maximum Temperature

area of 4 bottom tiers. Then, when the memory utilization ratio is increased to 95% (the left second column), less memory space is left. The improvement of our method is decreased to 14.0°C and 12.3°C. At the mean time, the temperature of M_1 and M_2 is only slightly increased because it is dominated by the worst case.

Next experiment is to increase the clock rate of the processor from 800 MHz to 1.2 GHz. In general, this will increase the access frequency of each *segment* due to the increased throughput. When the clock rate of processor is 1.2 GHz and utilization 75% (the right second column), the temperature of all mappings is increased by 7.7°C in average. Notice that the increase in temperature by our method is larger than those of M_1 and M_2. The reason is as follows. For *segments* which are accessed with high frequency, the processor needs to be stalled frequently for memory access. This means the memory *segment* is accessed at a near saturated frequency and increasing the clock rate of processor will only leads to limited increase in access frequency. However, for *segments* accessed with low frequency, the increase in access frequency will be proportional to the increase ratio of processor's clock rate. Since maximum temperature is usually observed on tiers with less heat dissipating ability and M_ours maps *segments* with low access frequency to these tiers, access frequency of these tiers is increased significantly as compared to other tiers. Therefore, M_ours cannot provide the same temperature reduction when clock rate of processor is increased. Still, our method reduces the temperature by 14.4°C and 12.0°C as compared to M_1 and M_2 (75%, 800 MHz). Finally, when the clock rate of processor is set to 1.2 GHz under 95% memory utilization (the right most column), the temperature is reduced by 12.1°C and 9.4°C as compared to M_1 and M_2 (95%, 800 MHz). Also, notice that in all experiments, M_2 is consistently better than M_1, which confirms our observations to form dies in adjacent tiers in a *group*.

VI. CONCLUSION

In this paper, we have proposed a static thermal management scheme for DRAM dies in stacked 3D designs. Both physical and software level issues are considered in our method. In physical level, the floorplan of DRAM die and power behavior of bank access are analyzed to generate candidate *configurations*. In software level, the memory space of the programs run on the system are partitioned to *segments* based on access frequency. The *configuration* decision and the mapping *segments* to physical locations are formulated as an ILP problem. Experiments show that our method can reduce temperature of memory system by 17.2°C as compared to a straightforward mapping in best case, and 13.4°C in average.

REFERENCES

[1] K. L. Tai, "System-In-Package (SIP): Challenges and Opportunities," *Asia and South Pacific Design Automation Conference,* pp. 191-196, 2000.

[2] Alexandru Pancescu, "Hynix Storms The NAND Industry - 24 nand memory chips only 1.4mm thick," *SOFTPEDIA,* Sep. 7, 2007.

[3] K. L. Tai, R. C. Frye, B. J. Han, M. Y. Lau, and D. Kossives, "A chip-on-chip DSP/SRAM multichip module," *Int'l Conf. on Multi-chip Modules,* pp 466-471, 1995.

[4] Y. L Low, R.C Frye, and K. J OConner, "Design methodology for chip-on-chip applications," *IEEE Trans. on Components, Packaging, and Manufacturing Technology* Part B, vol. 21, pp. 298-301, Aug. 1998.

[5] M. X. Wang, K. Suzuki, W. W.-M. Dai, Yee L. Low, K. J. Oconner and K. L. Tai, "Integration of Large-Scale FPGA and DRAM in a Package Using Chip-on-Chip Technology", *Asia and South Pacific Design Automation Conference,* pp. 205- 210, 2000.

[6] Michael Wang, Katsuharu Suzuki, Wayne Dai, Atsushi Sakai, Kiwamu Watanabe, "Configurable Area-IO Memory for System-in-a-Package (SiP)," *27th European Solid-State Circuits Conference,* September, 2001.

[7] Michael Wang, Katsuharu Suzuki, Wayne Dai, "Memory and Logic Integration for System-in-a-Package," *4th Int'l Conf. on ASIC*, October, 2001.

[8] Kiran Puttaswamy and Gabriel H. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," *ACM/IEEE Great Lakes Symposium on VLSI,* pp 19-24, 2006.

[9] Y. I. Kim, K. H. Yang, W. S. Lee, "Thermal Degradation of DRAM Retention Time: Characterization and improving techniques," *Proceedings of the 42nd IEEE Int'l Reliability Physics Symp.,* pp. 667-668, April 2004.

[10] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture,* February 2001.

[11] K. Skadron, T. Abdelzaher and M. R. Stan, "Control Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture,* February 2002.

[12] Y. Li, D. Brooks, Z. Hu, and K. Skadron, "Performance, Energy, and Thermal Considerations for SMT and CMP architectures," *Proceedings of the 8th Int'l Symp. on High-Performance Computer Architecture,* Feb. 2005.

[13] K. Sankaranarayanan, S. Velusamy, M.R. Stan, and K. Skadron, "A Case for Thermal-Aware Floorplanning at the Microarchitectural Level," *The Journal of Instruction-Level Parallelism,* Septempter 2005.

[14] M. Mutyam, F. Li, V. Narayanan, M. Kandemir and M. J. Irwin, "Compiler-Directed Thermal Management for VLIW Functional Units," *In. ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and. Tools for Embedded Systems,* June 2006.

[15] Y-F. Tsai, Yuan Xie, N. Vijaykrishnan, M. J. Irwin, "Three-Dimensional Cache Design Exploration Using 3DCacti," *Proceedings of IEEE International Conference on Computer Design (ICCD),* pp. 519-524, Oct. 2005.

[16] W. Huangry, K. Sankaranarayanany, R. J. Ribandoz, M. R. Stan and K. Skadron, "An Improved Block-Based Thermal Model in HotSpot 4.0 with Granularity Consideration", *Proceeding of the Workshop on Duplicating, Deconstructing, and Debunking,* June 2007

[17] C. Lee, M. Potkonjak and W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," in *30th MICRO,* pp. 330-335, December 1997

[18] A. Malik, B. Moyer and D. Cermak, "A Lower Power Unified Cache Architecture Providing Power and Performance Flexibility," *International Symposium on Low Power Electronics and Designs,* 2000

[19] *JM H.264/AVC CODEC 14.1,* http://iphome.hhi.de/suehring/tml/

[20] D. C. Burger, T. M. Austin and S. Bennett, "Evaluating Future Microprocessors– The SimpleScalar Tool Set," Technical Report 1342, University of Wisconsin-Madison, CS Department, June 1997

[21] *lp_solve,* http://lpsolve.sourceforge.net/