Segment-Based Task Scheduling for Thermal Optimization of Stacked Memory Architecture

WEI-KAI CHENG, TING-WEI HSU, RUEY-YEU WANG
Department of Information Computer Engineering
Chung Yuan Christian University
Chung Li, Taiwan 32023
wkcheng@cycu.edu.tw, wadekyowade@hotmail.com, g10377013@cycu.edu.tw

Abstract: - Heterogeneous integration enabled by 3D technology is one of the innovations for future microprocessor design. The heterogeneous integration of DRAM and multi-core processor in the 3D architecture offer much higher memory bandwidth, and mitigating the memory wall problem in off-chip DRAM design. However, stacking of multiple memory tiers comes out a serious thermal problem. In this paper, we propose a segment-based task scheduling methodology for this stacked memory architecture, and resolve this problem by ILP formulations. The proposed approach is integrated with task allocation and memory mapping in our system. Experimental results from the thermal simulation tool show that the proposed segment-based approach can reduce the thermal temperature by about 10% than using the task scheduling approach directly.

Key-Words: - 3D architecture, TSVs, memory mapping, task scheduling

1 Introduction

In comparison with two dimensional integrated circuits, 3D ICs stack multiple dies in the vertical direction, and transfer data between stacking dies by the high density through silicon via (TSVs). The architecture of 3D ICs provide the benefits of reducing wire length, communication delay, and power consumption on connection wires, increasing communication bandwidth, minimizing the overall package size, and heterogeneous integration of devices with different semiconductor processes.

There have been many researches on the design and synthesis of multi-core processors and memories targeted at the 3D ICs architecture. Research [1] implemented a checking core to improve the reliability while not increasing too much hardware cost. In research [2], micro architecture techniques are applied to control hotspots in the 3D integrated circuits. Researches [7, 9] analyzed in detail the benefit of performance efficiency by stacking memory, cache, and processor in the 3D architecture. Research [8] compared the tradeoff of different memory architecture in integrating with microprocessor. Research [15] explored true 3D DRAM organization to make better use of die to die memory bandwidth. In research [12], they investigated the degradation of DRAM retention time caused by thermal stress and the corresponding reliability problem. Research [10] proposed the concept of system in package (SIP) to reduce the overall cost while still match or close to the performance of system on chip (SOC). Research [11] discussed the 3D integration technology, EDA tools that enable the adoption of 3D ICs, and implementation of various microprocessor architectures. Research [16] surveyed various approaches to design 3D microprocessors, with their intrinsic ability to reduce the wire length, increase the memory bandwidth, and heterogeneous integration for innovation designs. Research [17] proposed a system level hardware/software co-synthesis framework for the 3D SOC design, including resource allocation, 3D layer partitioning and floorplanning, task scheduling and mapping, and evaluation of power, thermal, etc.

However, because of direct dies stacking, 3D ICs has the side effect of high power density and poor heat dissipation, result to the increase of overall temperature. For an integrated circuit to operating in the high temperature environment, it paid the cost of circuit reliability, performance efficiency, IR drop, die yield, and extra cooling system. Therefore, thermal issue has become a critical challenge in the design of 3D ICs.

Most of the thermal-aware optimization problem targeted on the stacked processor cores. Research [4, 5] controlled and optimized the circuit temperature through task scheduling. Not only for homogeneous multi-core architecture, had research [5] also optimized the thermal issue for the heterogeneous multi-core architecture. Research [14] proposed a quick and accurate mathematical model for task scheduling to optimize the system performance under the thermal constraint by clock frequency adaption. Research [3] proposed a set of mathematical equations for task load calculation, thermal temperature estimation, and management of power mode transition, such that they can consider thermal effect of 3D architecture effectively. In research [6], they create a thermal estimation model for 3D integrated circuits, and proposed a rotation scheduling methodology to reduce the peak temperature.

On the other hand, as memory occupies about 40% of power consumption and this percentage still increases continuously, thermal optimization on the stacked memory tiers become a serious problem in the future. Research [13] proposed a thermal-aware memory mapping technology by considering the behavior of power consumption and relative location of memory blocks simultaneously. However, research [13] targeted their problem on a fixed task execution order, the effect of task scheduling on thermal dissipation was not considered in their methodology.

In this paper, after task allocation and memory mapping, we propose a segment-based task scheduling methodology for thermal optimization on the 3D stacked memory architecture, and model this problem as ILP formulations. The contributions of this paper are summarized as follows:

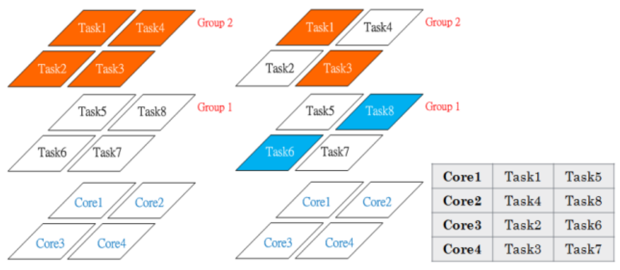
- We integrate task allocation, memory mapping, and task scheduling on the stacked memory architecture.
- We propose a segment-based task scheduling approach to avoid intensive memory access on the same memory group continuously for thermal optimization.
- ILP formulations are proposed for the segment-based task scheduling and idle slot allocation problems.

The rest of this paper is organized as follows. Section 2 describes the proposed 3D architecture and the motivation of segment-based task scheduling. Section 3 shows the problem formulation and system flow. In Section 4, we describe the proposed ILP formulations for segment-based task scheduling. Experimental results in Section 5 show the thermal effect of our segment-based task scheduling algorithms. Finally, we draw the concluding remarks in Section 6.

2 Motivation and Target Architecture

2.1 Task Allocation and Memory Mapping

We optimize the thermal-aware tasks execution on the stacked memory architecture in three aspects. The first is task allocation to balance the memory accesses of processor cores. The second is memory mapping to keep intensively memory access banks closer to heat sink and away from processor cores. And the final is task scheduling to avoid intensive memory access on the same memory group continuously. For the example in Fig. 1, suppose Task1, Task2, Task3, and Task4 are memory bound tasks, while Task5, Task6, Task7, and Task8 are CPU bound tasks. In the first stage, each processor core is allocated with a memory bound task and a CPU bound task to balance their power consumption and memory accesses. Then, in the memory mapping stage, all memory bound tasks access memory from Group2 that far away from the processor cores for better thermal dissipation; and all CPU bound tasks access memory from Group1. Finally, in the task scheduling stage, if {Task1, Task4, Task2, Task3} are scheduled to execute on {Core1, Core2, Core3, Core4} simultaneously, all dies in memory Group2 will be accessed intensively as shown in Fig. 1(a), hence increasing the overall temperature of memory tiers instantly. However, if {Task1, Task8, Task6, Task3} are scheduled to execute on {Core1, Core2, Core3, Core4} simultaneously, memory accesses are as shown in Fig. 1(b), temperature of the memory tiers could remain stable.



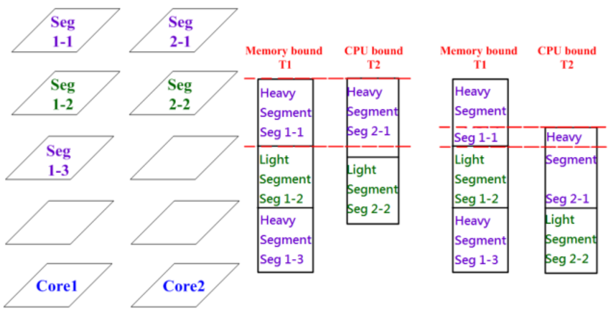
Core1	Task1	Task5
Core2	Task4	Task8
Core3	Task2	Task6
Core4	Task3	Task7

(a) (b)

Fig. 1: Task scheduling to balance memory loading.

2.2 Segment-based Task Scheduling

Although the task scheduling in Fig. 1(b) can avoid the execution of memory bound tasks simultaneously, a CPU bound task still may have certain code segments access memory intensively. Therefore, we propose a segment-based task scheduling methodology for thermal optimization of 3D stacked memory and processor architecture. For the example in Fig. 2, a memory bound task Task1 is partitioned into three page segments, and a CPU bound task Task2 is partitioned into two page segments. Among which, Seg1-1, Seg1-3 and Seg2-1 are memory intensive segments (heavy segments), while the other two are memory non-intensive segments (light segments). If segments are scheduled as shown in Fig. 2(a), Seg1-1 and Seg2-1 are totally overlapped such that Core1 and Core2 will access memory intensively in the same period of time. However, if sub-segments are scheduled as shown in Fig. 2(b), Seg1-1 and Seg2-1 are overlapped with only a little of time, which will benefit the thermal behavior of stacked processor cores and memory banks.



(a) (b)

Fig. 2: Segment-based task scheduling to balance memory loading.

2.3 Target Architecture

Fig. 3 shows our target architecture. We suppose there are four cores in a processor tier, stacked on it are memory tiers, and heat sink is on top of memory tiers. In each memory tier, there are four memory dies. And in each memory die, there are four memory banks. Because we restrict the available memory size and balance the memory accesses of each processor core during the task allocation phase as described in Section 3, we make the assumption that each processor core can only accesses the memory dies vertically stacked on top of it through TSVs.

In order to compare with research [13] fairly, we also suppose that only one memory bank can be accessed at a time for each memory die. Therefore, if the width of system data bus is a multiple of the bandwidth of a memory bank, a multiple of vertically stacked memory dies need to be accessed simultaneously. For this reason, we partition the memory tiers into groups in the vertical direction. For the example, if width of the system data bus is 32 bits, and the bandwidth of a memory bank is 8 bits, four memory tiers will form a group as shown. If there are M memory tiers in the architecture, and N memory tiers form a group, then there will be totally M/N groups. The parameters M and N are all configurable in our target architecture.

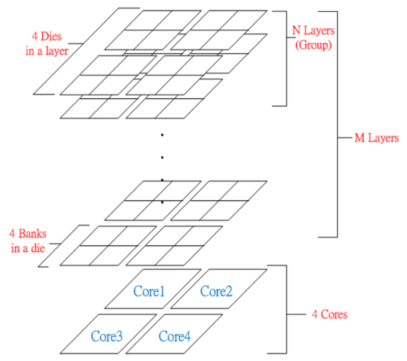


Fig. 3: Proposed target architecture.

3 Algorithms and System Flow

Fig. 4 shows our system flow. The first phase is segments generation, we use the program analysis tool Simplescalar3.0 [19] to get the relative information for instruction count, execution time, memory address, and reference count of memory.

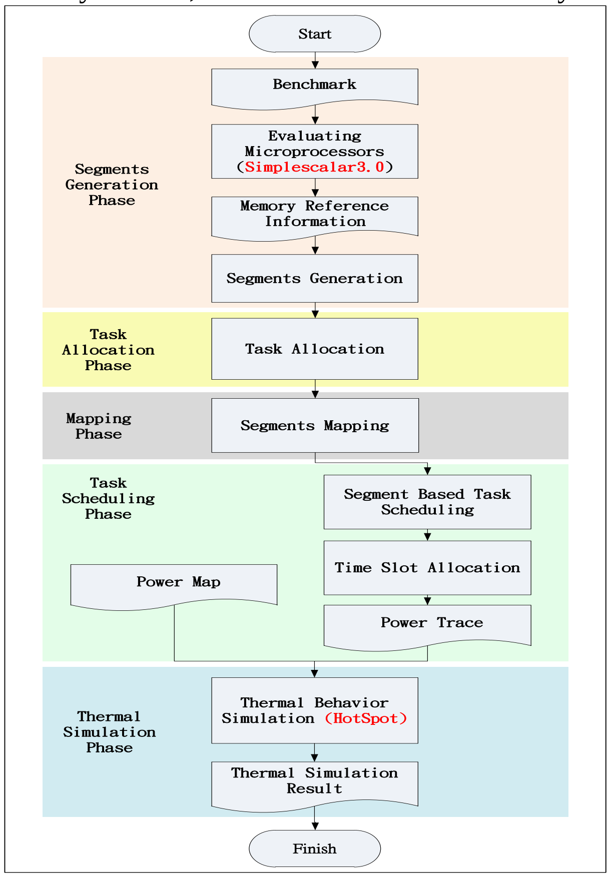


Fig. 4: System Flow.

According to the reference address of memory and the flow of program execution, we partition the application programs into data segments to fit the size of memory banks as the pseudo code shown in Fig. 5. The first step is segment classification based on the address of memory reference as shown in line 1 to line 10. If a memory reference does not belong to any created data segment, a new data segment is created. The second step is to merge data segments to fit the size of memory bank as shown in line 12, data segments are merged in the order created in the first step. And the third step is segments reconstruction to balance the size of data segments, and update the reference record of data segments as shown in line 13.

The second phase is task allocation. We balance both memory size requirement and memory references among the processor cores. The approach in research [1] is applied to balance power consumption, and we modify it in our work to take into account the memory references issue simultaneously.

Under the memory size constraint of memory dies stacked on each processor core, we firstly classify and sort the tasks based on their memory size requirement. In the case when two or more tasks belong to the same class of memory size requirement, they are sorted based on their memory references. Then we allocate the sorted tasks to processor cores sequentially to balance both their memory size and memory references.

```
Algorithm : Segment Generation

Input : Memory reference record
Output : Memory segments

1.  While ( end of reference record )
2.  {
3.      ref = GetNextReference( );
4.      seg = FindRefInSegment( );
5.
6.      If ( seg == NULL )
7.          CreateNewSegment( ref );
8.      Else
9.          RefreshInfo( seg, ref );
10. }
11.
12. MergeSegments( );
13. UpdateSegments( );
```

Fig. 5: Pseudo code of segment generation.

Fig. 6 shows an example to illustrate the effect of this two-stage sorting approach. In this example, we suppose there are totally eight tasks to be allocated to four processor cores, each task has its memory size requirement and memory references, and the memory size stacked on each processor core is eight memory segments. The result of task sorting by only memory references is {Task1, Task5, Task6, Task2, Task3, Task4, Task8, Task7}, Fig. 6(a) shows the result when we allocate tasks in sequential based on this sorting order to balance memory references of processor cores. The allocation result shows that memory size requirement of core 2 exceeds the memory segments stacked on it, and hence needs to access the memory segments stacked on other processor cores in some time, resulting to the bad thermal dissipation. While if our two-stage sorting approach is applied, the result of task sorting is {Task1, Task6, Task5, Task2, Task4, Task7, Task3, Task8}, Fig. 6(b) shows the result when we allocate tasks in sequential based on this sorting order to balance memory references of processor cores. The allocation result shows that no memory requirement of processor core exceeds the memory segments stacked on it, and the memory references of processor cores is even better balanced than that in Fig.6(a). Therefore, with this task allocation approach, we make the assumption in our target architecture that each processor core can only accesses the memory dies vertically stacked on top of it as described in Section 2.3.

Task	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Total Seg Num	8	3	1	3	3	6	3	1
Total Frequency	20	8	7	6	15	9	3	4

	Tasks	Total Segments	Total Frequency
Core 1	T1	8	20
Core 2	T3,T6,T7	10	19
Core 3	T5	3	15
Core 4	T2,T4,T8	7	18

(a)

	Tasks	Total Segments	Total Frequency
Core 1	T1	8	20
Core 2	T3,T6	7	16
Core 3	T5,T7	6	18
Core 4	T2,T4,T8	7	18

(b)

Fig. 6: Example of two-stage sorting approach for task allocation.

The third phase is memory mapping. We apply and modify the approach proposed in research [13] to fit our target architecture. The first step is to avoid the accesses of memory banks in the vertical direction simultaneously, which has been proved to have critical impact on the thermal behavior of 3D memory architecture in research [13]. In the second step, we further consider the thermal effect of memory mapping in the same memory tier by memory banks classification.

As shown in Fig. 7, there are four memory dies in a memory tier, and four memory banks in a memory die. The memory banks marked with region 1 locating in the corner and neighbouring with less memory banks, hence have the best thermal dissipation. On the other hand, the memory banks marked with region 3 locating in the centre and neighbouring with more memory banks, hence have the worst thermal dissipation. Therefore, memory segments with high access frequency should be mapped to region 1 to reduce thermal temperature, and mapped to region 3 in the versa.

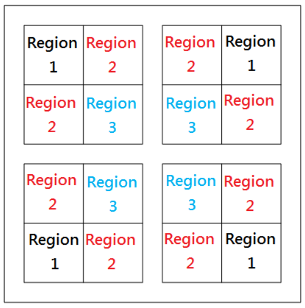


Fig. 7: Classification of memory banks in terms of thermal dissipation in the same memory plane.

Based on this observation, we map task segments to memory banks by two methods. As illustrated in Fig. 8, we suppose there are four tasks executing on the same processor core and access the stacked bottom-right corner of memory tiers. If memory access frequencies of task3 and task4 are much higher than that of task1 and task2, we apply the first memory mapping method as shown in Fig. 8(a), in which memory-bound tasks access memory banks far from the centre of memory tiers to have better thermal dissipation. On the other hand, if memory access frequencies of all the four tasks are not obviously different, we apply the second memory mapping method as shown in Fig. 8(b), in which all tasks share the centre memory banks in even.

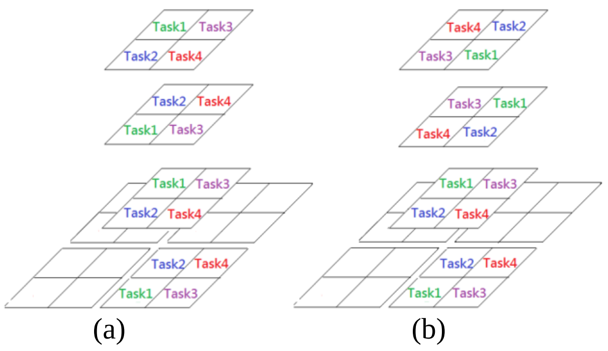


Fig. 8: (a) Memory mapping for tasks with non-even memory access frequencies. (b) Memory mapping for tasks with even memory access frequencies.

Finally, under the constraint of finished deadline of tasks, ILP formulations for segment-based task scheduling and time slot allocation are proposed to minimize the time overlap of massive memory references of processor cores, such that we can optimize the thermal behavior of multi-core processor and the stacked memory tiers on it. Detail description and ILP formulations of segment-based task scheduling and time slot allocation are described in the next section.

4 Segment-Based Task Scheduling

4.1 Segment-Based Task Scheduling

In the segment-based task scheduling, we firstly partition the segments accessed by a task into sub-segments according to its program structure and execution behavior. For the example in Fig. 9, there are two tasks allocated to core N, and both tasks access two memory segments. For task1, the two memory segments seg1-1 and seg1-2 are further partitioned into five sub-segments {sub1-1-1, sub1-1-2, sub1-1-3, sub1-2-1, sub1-2-2}; and for task2, two memory segments seg2-1 and seg2-2 are further partitioned into four sub-segments {sub2-1-1, sub2-1-2, sub2-2-1, sub2-2-2}. Sub-segments sub1-1-3 and sub2-1-1 are duplicated because of multiple accesses in different stages of their task execution. An edge between two sub-segments implies the execution order constraint in the task. After sub-segments duplication, there are totally eleven sub-segments and two dependency streams in this example.

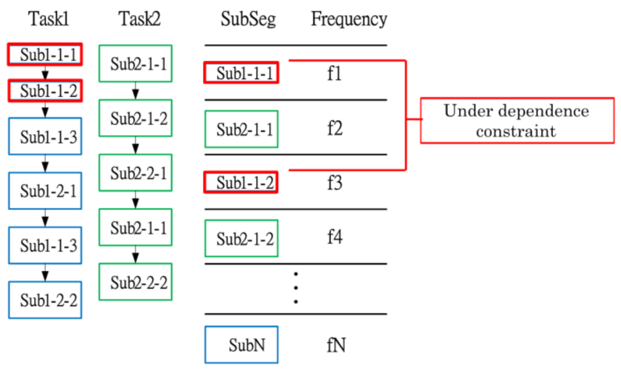


Fig. 9: Example of segment-based task scheduling.

There is execution order constraint for sub-segments that belong to the same task. However, there is no execution order constraint for sub-segments that belong to different tasks. For the example in Fig. 9, there is an execution order constraint between sub-segments sub1-1-1 and sub1-1-2, but there is no execution order constraint between sub-segment sub2-1-1 and any sub-segment of task1. In this paper, we define the segment-based task scheduling problem as to determine the execution order of all sub-segments allocated to the same processor core under the execution order constraint. We define the objective function to maximize the total difference of access frequencies between sub-segments that scheduled in successive execution order, such that sub-segments with high memory access frequency and sub-segments with low memory access frequency are shuttled to avoid intensively memory access in continuous time.

To get the best result, we model the segment-based task scheduling problem by integer linear programming (ILP) formulations, and apply this methodology to all the processor cores. Notations, objective function, and constraints of these formulations are described as below.

- Notations

S_i: sub-segment *i*

SN: total number of sub-segments after duplication in the target processor core

feq_i: memory access frequency of sub-segment *i*

$x_{i,j}$: a binary number to represent whether sub-segment *i* is scheduled in the *j*th execution order

o_i: an integer number to represent the execution order of sub-segment *i*

- Objective function

We define the objective function of the proposed segment-based task scheduling as to maximize the total difference of memory access frequencies between all successive execution sub-segments. The larger the value of this objective function, the less opportunity a memory bank will be accessed intensively in continuous time. Formulation of the objective function is described as below.

$$\text{MAX}: \sum_{j=1}^{SN}\sum_{i1=1}^{SN}\sum_{i2=1}^{SN} \left| feq_{i1} - feq_{i2} \right| * x_{i1,j} * x_{i2,j+1}$$

- Constraints

1. Each sub-segment can be scheduled to only one execution order.

$$\sum_{j=1}^{SN} x_{i,j} = 1, \text{ for each subsegment } i$$

2. When sub-segment *l* is dependent on sub-segment *i*, its execution order must be later than sub-segment *i*.

$$o_l \geq o_i + 1 \text{ for } S_i \rightarrow S_l$$

$$\text{where } o_i = \sum_{j=1}^{SN} j * x_{i,j}$$

3. Each execution order can have only one sub-segment scheduled to it.

$$\sum_{i=1}^{SN} x_{i,j} = 1, \text{ for each execution order } j$$

4.2 Time Slot Allocation

Idle slots are inserted between sub-segments to release the thermal pressure of processor cores due to continuously task execution and memory access. Fig. 10 shows the example of time slot allocation, in Fig. 10 (a) and Fig. 10 (b), one time slot is allocated at the end and the beginning of ASAP scheduling and ALAP scheduling, respectively. As shown in Fig. 10 (c), the time slot allocation problem is defined as to find an idle slots insertion solution between ASAP scheduling and ALAP scheduling of sub-segments, such that to get the maximal thermal reduction.

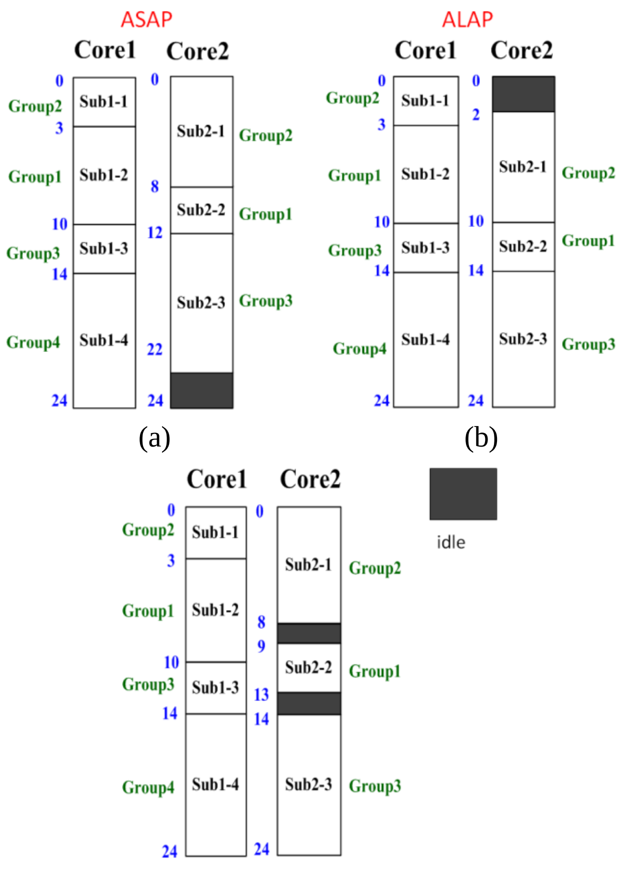


Fig. 10: Time slot allocation of sub-segments and idle slots.

Since memory segments with intensively access frequency are map to the memory groups that close to the heat sink, and we aim at avoiding accessing these memory groups simultaneously, cost function is designed to insert idle slots such that the time overlapping of sub-segments that access the same memory group is as less as possible.

The example for evaluation of time slot allocation is shown in Fig. 11. Suppose sub-segment sub1-2 is scheduled in time slot 3, and sub-segment sub1-3 is scheduled in time slot 10. If sub-segment sub2-2 of Core2 is scheduled in time slot 8, its execution time overlaps with both the sub-segments sub1-2 and sub1-3 of Core1, and the overlapping time are all two time slots. In this example, sub-segments sub1-2 and sub2-2 all have more memory accesses and are mapped to memory Group1, while sub-segment sub1-3 has less memory accesses and is mapped to memory Group3.

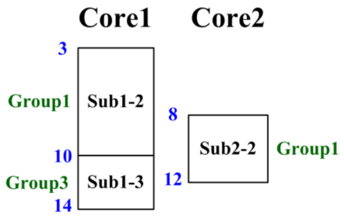


Fig. 11: Evaluation of time slot allocation.

As shown in Fig. 11, time overlapping of sub-segments sub1-2 and sub2-2 is not preferred because they all have intensive memory accesses. Since we aim at avoid all processor cores accessing the same memory group simultaneously as illustrated in Fig. 1 and Fig. 2, cost function of this time overlapping is defined as below.

(Total Group Number - | Group of Sub1-2 - Group of Sub2-2 |) * (#Overlapping Time Slots)

Suppose totally there are four memory groups, the cost of time overlapping for sub-segment sub2-2 with sub-segments sub1-2 and sub1-3 is calculated as (4 -|1-1|) * 2 + (4 - |3-1|) * 2 = 12. The less the cost is, the better the time slot a sub-segment should be scheduled to. To achieve this objective function, we propose an integer linear programming (ILP) formulation for this time slot allocation problem. Notations, objective function, and constraints of these formulations are described as below.

- Notations

Cycle: total number of time slots
Core: total number of processor cores
GN: total memory group number
G_i: memory group that sub-segment i is mapped to
S_i: sub-segment i
SN_c: total number of sub-segments after duplication in the target processor core c
len_i: execution length of sub-segment i in terms of time slot number
$x_{i,j}$: a binary number to represent whether sub-segment i is scheduled in the jth time slot
b_i: an integer number to represent the beginning time slot of sub-segment i
e_i: an integer number to represent the ending time slot of sub-segment i
$overlap_{j,c}$: cost of time overlapping of sub-segments in the jth time slot between processors cores c and c+1

- Objective function

We define the objective function as to minimize the time overlapping of sub-segments with intensive memory access among different processor cores. The smaller the value of cost function, the less opportunity two memory banks in the same group will be accessed intensively in the same time. Formulation of the objective function is described as below.

$$\text{Min}: \sum_{c=1}^{\text{Core-1}} \sum_{j=1}^{\text{Cycle}} \text{overlap}_{j,c}$$

where

$$\text{overlap}_{j,c} = (GN - |G_{i1} - G_{i2}|) * x_{i1,j1} * x_{i2,j2}$$
$$i1 \in \text{segment run on core } c,$$
$$j1 = b_{i1} \text{ and } b_{i1} \le j \text{ and } e_{i1} \ge j,$$
$$i2 \in \text{segment run on core } c+1,$$
$$j2 = b_{i2} \text{ and } b_{i2} \le j \text{ and } e_{i2} \ge j$$

- Constraints

1. Each sub-segment can be scheduled to only one time slot

$$\sum_{j=1}^{\text{Cycle}} x_{i,j} = 1, \text{ for each sub-segment } i$$

2. When sub-segment l is dependent on sub-segment i, beginning time slot of sub-segment l must be later than ending time slot of sub-segment i

$$e_i < b_l \text{ for } S_i \rightarrow S_l$$

where

$$e_i = \sum_{j=1}^{\text{Cycle}} (j + len_i - 1) * x_{i,j}$$

$$b_l = \sum_{j=1}^{Cycle} j * x_{l,j}$$

3. For each processor core *c*, each time slot can have at most one sub-segment scheduled to it

$$\sum_{i=1}^{SNc} x_{i,j} \le 1, \text{ for each time slot } j$$

5 Experimental Results

We implement our system in the C/C++ programming language, and use Extended LINGO Release 11.0 as the ILP solver for the proposed segment-based task scheduling and time slot allocation problems. The platform is Windows-7 x64 running on i5-2500K quad-cores CPU with 4GB RAMs.

Table 1
System parameters

System Parameter	**Value**
Total tiers	9
Memory tiers	8
Processor tier	1
Dies on memory tier	4
Banks per memory tier	16
Bit width of a memory die	8
Bit width of system bus	32
DRAM Parameter	**Value**
Capacity per die	256 Mb
Total Memory Size	1GB
Processor Parameter	**Value**
Processor	Alpha
Cores number	4

We test our system flow and segment-based approach by four test benches, each test bench is consist of a set of application programs from SPEC [18] and MiBench [21], and use program analysis tool Simplescalar3.0 [19] to get the information for instruction count, execution time, and reference count of memory. Power consumption of memory is calculated based on the Micron DDR3 specification [23], the processor cores in our architecture are Alpha 21264 [22]. For all the experimental results, we use thermal simulator HotSpot [20] to simulate and report the highest temperature, the lowest temperature, and the average temperature in the 3D floorplanning. Table 1 lists the parameters of our target system.

Table 2 compares the effect of different task allocation approaches. The notation BF denotes the task allocation approach based on memory access frequency only, and our two-stage sorting approach considers both memory access frequency and memory size requirement. Both approaches all apply our memory mapping and segment-based task scheduling methodologies. Experimental results show that our approach gets lower thermal temperature on all the three corners.

Table 3 compares the effect of different memory mapping approaches. The direct mapping method does not avoid the memory access in the vertical direction; while our memory mapping approach not only avoids accessing memory in the vertical direction, but also considers the thermal effect of memory mapping in the same memory tier. Both approaches all apply our task allocation and segment-based task scheduling methodologies. Experimental results show that our memory mapping approach improves the thermal temperature over ten percentage for all the test benches.

Table 4 shows the effect of our segment-based task scheduling and time slot allocation methodologies. With the same task allocation and memory mapping results by our methodologies, experimental results of our segment-based approaches further improve the thermal temperature by about ten percentages for all the test benches than that using the task scheduling directly.

Comparing Table 2, Table 3 and Table 4, experimental results show that memory mapping and segment-based task scheduling have larger impact on the thermal temperature than task allocation. Therefore, we further compare the effect of memory mapping and segment-based task scheduling as the results shown in Table 5. When both memory mapping and segment-based task scheduling approaches are all not applied, the results are the worst. The results of applying memory mapping approach only are a little better than that of applying segment-based task scheduling approach only, this is because that vertical direction has larger impact on the thermal temperature. However, when both approaches are applied, there is still obvious thermal temperature improvement over the memory mapping approach only. Therefore, although the segment-based task scheduling approach is not the unique factor to reduce thermal temperature, applying it together with memory mapping can avoid the continuous access of memory hot spot and further improve the thermal temperature.

Table 2
Thermal results of task allocation approaches

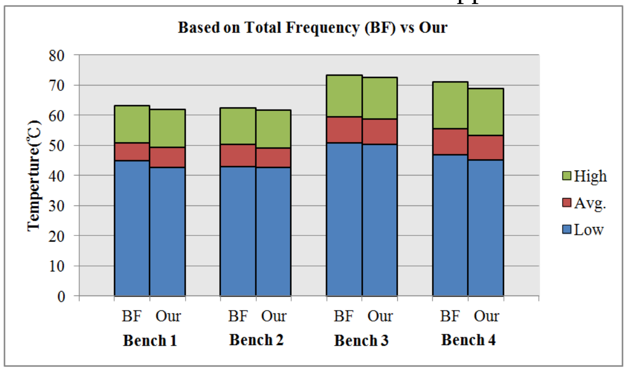


Table 3
Thermal results of memory mapping approaches

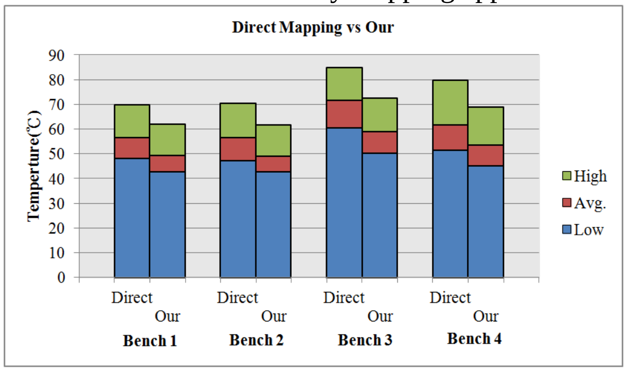


Table 4
Thermal results of task scheduling approaches

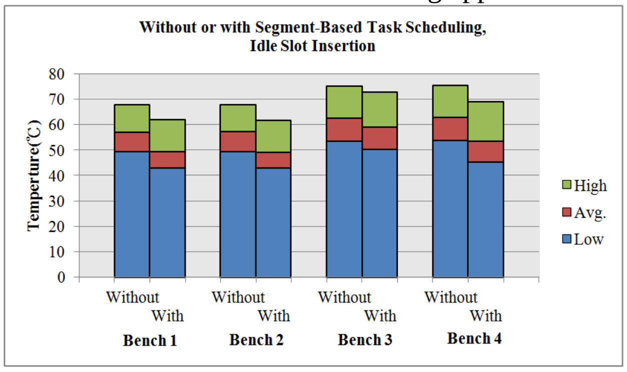
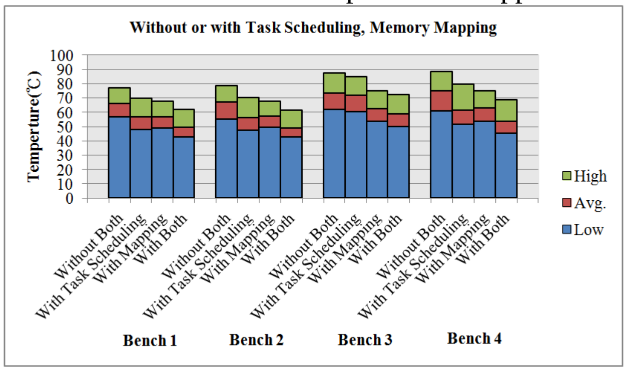


Table 5
Thermal results of different optimization approaches



6 Conclusions

In this paper, we propose a methodology for thermal optimization of 3D stacked memory architecture. In addition to task allocation and memory mapping, ILP formulations for segment-based task scheduling and time slot allocations are proposed to further reduce the thermal temperature of stacked memory architecture. Experimental results from the HotSpot thermal simulator show that our approach is indeed effective for this optimization problem.

7 Acknowledgement

This work was supported in part by the National Science Council of Taiwan, R.O.C., under grant number NSC 101-2221-E-033-069.

References:

[1] N. Madan and R. Balasubramonian, "Leveraging 3D Technology for Improved Reliability", *40th International Symposium on Microarchitecture, 2007.*

[2] K. Puttaswamy and G. H. Loh, "Thermal Herding: Microarchitecture Techniques for Controlling HotSpots in High-Performance 3DIntegrated Processors", *13th International Symposium on High Performance Computer Architecture, 2007.*

[3] Changyun Zhu, Zhenyu Gu, Li Shang, R.P. Dick and R. Joseph, "Three-Dimensional Chip-Multiprocessor Run-Time Thermal Management", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.27, no.8, pp. 1479-1492, August 2008.*

[4] Han Wang, Yuzhuo Fu, Ting Liu and Jiafang Wang, "Thermal management via task scheduling for 3D NoC based multi-processor", *International SoC Design Conference (ISOCC), pp.440-444, November 2010.*

[5] Xiuyi Zhou, Jun Yang, Yi Xu, Youtao Zhang and Jianhua Zhao, "Thermal-Aware Task Scheduling for 3D Multicore Processors", *IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 1, pp. 60-71, January 2010.*

[6] Jiayin Li, Meikang Qiu, Jianwei Niu, Tianzhou Chen and Yongxin Zhu, "Real-Time

Constrained Task Scheduling in 3D Chip Multiprocessor to Reduce Peak Temperature", *8th International Conference on Embedded and Ubiquitous Computing (EUC), pp. 170-176, December 2010.*

[7] G. L. Loi, B. Agarwal, N. Srivastava, S.-C. Lin, and T. Sherwood, "A Thermally-Aware Performance Analysis of Vertically Integ*rated (3-D) Processor-Memory Hierarchy", 43rd Design Automation Conerence (DAC), 2006.*

[8] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the Processor-Memory Performance Gap with 3D IC Technology", *IEEE Design and Test of Computers, 22(6): pp. 556–564, Nov.-Dec. 2005.*

[9] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, and D. Pantuso, "Die stacking (3D) microarchitecture", *39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 469–479, 2006.*

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

[11] Y. Xie, G. Loh, B. Black, and K. Bernstein, "Design space exploration for 3D architectures", *ACM Journal of Emerging Technologies in Computing Systems, 2006.*

[12] Y. I. Kim, K. H. Yang and W. S. Lee, "Thermal Degradation of DRAM Retention Time: Characterization and improving techniques", *42nd IEEE International Reliability Physics Symposium, pp. 667-668, April 2004.*

[13] Ang-Chih Hsieh and TingTing Hwang, "Thermal-aware memory mapping in 3D designs", *Design, Automation & Test in Europe Conference & Exhibition (DATE), pp.1361-1366, April 2009.*

[14] Chiao-Ling Lung, Yi-Lun Ho, Ding-Ming Kwai and Shih-Chieh Chang, "Thermal-aware on-line task allocation for 3D multi-core processor throughput optimization", *Design, Automation & Test in Europe Conference & Exhibition (DATE), pp.1-6, March 2011.*

[15] G. H. Loh, "3d-stacked memory architectures for multicore processors", *International Symposium on Computer Architecture (ISCA), pp. 453-464, 2008.*

[16] Y. Xie, "Processor Architecture Design Using 3D Integration Technology", *23rd International Conference on VLSI Design, pp. 446-451, January 2010.*

[17] Q. Zou, Y. Chen and Y. Xie, A. Su, "System-level design space exploration for three-dimensional (3D) SoCs", *9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 385-388, October 2011.*

[18] http://www.spec.org/

[19] 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.*

[20] W. Huang, K. Skadron, S. Gurumurthi, R. J. Ribando, and M. R. Stan. "Differentiating the Roles of IR Measurement and Simulation for Power and Temperature-Aware Design", *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), April 2009.*

[21] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", *Proceedings of the Workload Characterization, 2001.*

[22] M. K. Gowan et al., "Power Considerations in the Design of the Alpha 21264 Microprocesso"r, *Design Automation Conerence (DAC), pp. 726-731, 1998.*

[23] http://www.micron.com