國 立 清 華 大 學

博 士 論 文

# 超大型積體電路設計下，針對功率、良率，及可靠度之架構探索

# Architecture Exploration for Power, Yield, and Reliability in VLSI Designs


所 別 資訊工程學系 組別 _____

學號姓名 944325 謝昂志 (Ang-Chih Hsieh)

指導教授 黃婷婷 博士 (Dr. TingTing Hwang)

中華民國一百年五月

# 摘　　要

隨著半導體製程技術的發展，先進的電子整合科技如系統級封裝技術(System in Package)及直通矽穿孔技術(Through Silicon Via)，讓設計者可以將更多的電子元件整合在單一設計中，並在極小的單位面積下，開發出在功能愈來愈強，複雜度愈來愈高的電子系統。這些先進的技術雖然可以帶來許多好處，卻也在半導體的設計及製造上，帶來了新的挑戰。在單位面積下，提升電子元件數量的同時，也會增加功耗密度(Power Density)，這將會導致熱能(Thermal)的問題，大幅影響現今電子系統的可靠度。因此，低功率的系統架構設計以及熱能管理機制在未來的電子系統中將是不可或缺的。此外，先進的整合科技中，需要用到較為複雜的製程技術，如何在使用這些技術的同時，維持高良率(Yield)，對於電子產業來說是非常重要的。在本論文中，針對快取(Cache)、主記憶體(Main Memory)，以及三維導線連結(3D Interconnect)等三個層級，我們提出了新的系統架構以及優化方法，以提升功率、可靠度，以及良率方面的表現。

首先，在快取記憶體系統的部份，我們開發了一個可以在動態下作調整(Run-Time Reconfigurable)的延展式快取架構(Expandable Cache)。由 G. Bournoutian 及 Orailoglu 所提出的延展式快取架構，可以在很小的面積成本下，有效的降低嵌入式系統(Embedded System)的快取誤失(Cache Miss)及功能消耗(Energy Consumption)。然而，原先的延展式快取架構，只使用了一種固定的延展方式，會產生嚴重的置換(Thrashing)問題。針對這個現象，我們以原先的延展式快取架構為基礎，開發出一個可以針對程式的動態執行特性，去變更延展方式的快取架構。透過執行在編譯(Compile)階段所加入的特殊組態指令(Configuration Instructions)，延展式快取架構的延展方式可以在動態下作變更。基於 SPEC 2000 的實驗結果，我們所提出的新快取架構，與原來的延展式快取架構相比，在快取誤失的比例上，有 14.74%的改善。在功能消耗的表現上，以雙路集合關聯(2-Way Set-Associative)快取架構為基準，我們所提出的架構比原先的延展式快取架構要好上 5.62%。

接下來，針對三維設計下的主記憶體系統，我們提出了一個考量熱能表現的記憶體映程(Memory Mapping)技術。在三維設計中，一個硬體記憶體區塊的熱能表現，取決於該區塊的功耗表現(Power Behavior)以及散熱能力(Heat Dissipating Ability)。功耗表現的部分，主要由所執行程式的存取特性來決定；而散熱能力，則要由記憶體區塊所在的實體位置所決定。故，一個考量熱能表現的記憶體配置(Memory Allocator)系統，必須有著下列兩項特性。第一，該配置系統要能同時考量一個記憶體區塊的功耗表現以及散熱能力；第二，該配置系統

要能考量到一個硬體記憶體區塊，在應用程式執行過程中的溫度變化。在本論文中，我們將會提出一個同時考量上述兩項特性的記憶體映成演算法。我們所提出的方法屬於靜態(Static)的熱能管理機制，主要運用於嵌入式系統軟體端的設計流程。實驗結果顯示，對於單核心的系統，我們的方法在最好的情形下，與傳統直接的記憶體映成方式相比，可將記憶體系統的最高溫度降低 17.1℃。平均來說，記憶體系統的最高溫度降低了 13.3℃。對於四核心的系統，當 L1 快取分別設為 4KB 與 8K 時，記憶體系統的最高溫度分別降低了 9.9℃與 11.6℃。

最後，針對直通矽穿孔導線的部分，我們提出了一套修復的機制。在三維設計中，直通矽穿孔導線，主要用於在垂直方向的訊號連接，對於三維晶片來說非常重要。但就像一般的半導體元件，直通矽穿孔導線的製造及連結，可能會因為製程上的問題而無法正常運作。當利用三維整合技術所堆疊的晶圓粒(die)愈來愈多時，因製程因素所造成的直通矽穿孔導線失效問題，將會嚴重的影響生產良率及製造成本。針對直通矽穿孔導線失效問題，本論文提出了一個合乎成本考量的修復架構。透過機率模型的分析，我們整理了一些重要的觀察結果。首先，當層與層(Tier-to-Tier)之間的直通矽穿孔導線數量少於 1000 及 10000 時，實際上層與層之間會失效的直通矽穿孔導線數量通常不大於 2 及 5。假設層與層之間失效的直通矽穿孔導線數量在 2 到 5 之間，足以涵蓋 99.9%所有的情形(包含有失效直通矽穿孔導線以及沒有的狀況)。在這樣的假設下，當每個直通矽穿孔導線區塊(TSV Block)各配置有一個冗於直通矽穿孔導線時，若將直通矽穿孔導線區塊所包含的導線數量限制在 50 及 25，可以達到 90%及 95%的修復率(對於有直通矽穿孔導線失效的情形來說)。就整體的良率來說，我們提出的修復架構，足以將絕大部分因直通矽穿孔導線瑕疵而失效的晶片加以修復，並將直通矽穿孔導線的生產良率提升到 99.4%。

# 誌　　謝

謹以此論文獻給

我的父母親-謝茂盛先生及李秋燕女士

我的指導教授-黃婷婷教授

我的口試委員-張世杰教授、王廷基教授、麥偉基教授、陳添福教授、

黃俊達教授

我的學長姐-一宇、文雯、武安、賢德、巽言、

昌博、岳叡、巧禎、侑儒

我的同學-柏元、名詔、世梁、偉恆、冠賢、福偉、博揚

我的學弟妹-哲宇、介俊、鈞澤、翌聖、

詠勝、俐宇、

俊成、宏隆、子誠、信皓、

安琪、家尉、俊宇、

明賢、宗祐、健銘

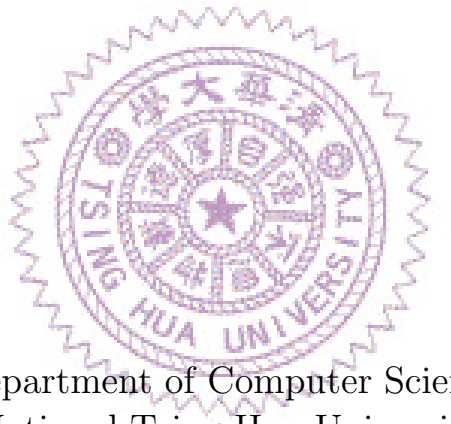
我的朋友-博耀、聿謙、仲韓、宗言、明釗、祐世、巧玲、光曜、勇志

And... My Beloved 家禎～

謝謝你們

# Architecture Exploration for Power, Yield, and Reliability in VLSI Designs

Student: Ang-Chih Hsieh
Advisor: TingTing Hwang




Department of Computer Science
National Tsing Hua University
HsinChu, Taiwan 300

May, 2011

**Abstract**

Modern integration technologies including system in package (SIP) and through silicon via (TSV) make it possible to put more devices in a single IC design. More powerful and complex systems can be developed within small areas. In addition to the advantages brought by these technologies, new challenges are revealed. Increasing the number of devices in a given area often results in higher power density. This causes thermal problems which can severely degrade the reliability of modern electronic systems. Low power architecture design and thermal management scheme are required. Also, more complex fabrication processes are required for modern integration technologies. How to improve the yield is a critical design issue for IC industry. In this dissertation, architecture designs and optimization technologies are proposed to improve power, reliability, and yield in cache, memory, and interconnect levels.

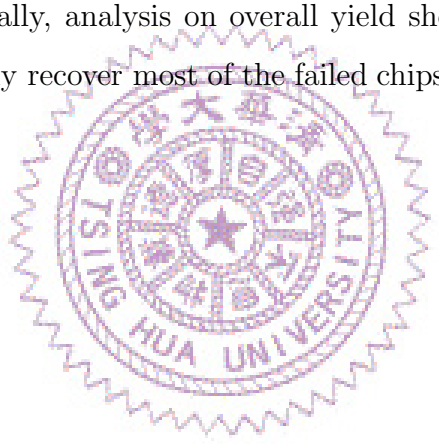
First, in cache level, a low power cache designed called *run-time reconfigurable expandable cache* is introduced. *Expandable cache* proposed by G. Bournoutian and Orailoglu is very efficient in reducing miss rate and energy consumption with small area overhead. However, the original *expandable cache* with only one expansion scheme may lead to thrashing problems. In this work, based on the structure of *expandable cache*, we will introduce a new cache design which has many expansion schemes to fit different run-time program behaviors. The expansion scheme of our proposed cache is dynamically changed by executing configuration instructions which are inserted at

compile time. The experimental results of *SPEC CPU2000* have shown that our proposed cache design effectively improves the miss rate by 14.74% as compared with the original *expandable cache*. In terms of energy improvement ratio, our method is 5.62% higher than that of *expandable cache* when the baseline is set as the energy consumption of 2-way set-associative cache.

Second, a thermal-aware memory mapping technique for 3D designs is proposed. DRAM is usually used as main memory for program execution. The thermal behavior of a memory block in a 3D design 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 logic block but also the physical location during memory mapping, second, the changing temperature of a physical block during execution of programs. In this thesis, 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, for single-core systems, our method can reduce the temperature of memory system by 17.1°C as compared to a straightforward mapping in the best case, and 13.3°C in average. For systems with 4 cores, the temperature reductions are 9.9°C and 11.6°C in average when L1 cache of each core is set to 4KB and 8KB, respectively.

Finally, the recovery of failed TSV is discussed. TSV provides commu-

nication links for dies in vertical direction and is a critical design issue in 3D integration. Just like other components, the fabrication and bonding of TSVs can fail. A failed TSV can severely increase the cost and decrease the yield as the number of dies to be stacked increases. A redundant TSV architecture with reasonable cost is proposed in this thesis. Based on probabilistic models, some interesting findings are reported. First, the number of failed TSVs in a tier is usually less than 2 when the number of TSVs in a tier is less than 1000 and less than 5 when the number of TSVs in a tier is less than 10000. Assuming that there are at most 2∼5 failed TSVs in a tier. With one redundant TSV allocated to one TSV block, our proposed structure leads to 90% and 95% recovery rates for TSV blocks of size 50 and 25, respectively. Finally, analysis on overall yield shows that the proposed design can successfully recover most of the failed chips and increase the yield of TSV to 99.4%.



# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Modern integration technologies including system in package (SIP) [23] and through silicon via (TSV) [44] [45] provide many benefits including high density, high-bandwidth, low power, and small form-factor [46]. With the capacity provided by these technologies, the number of devices integrated in a design is greatly increased. More powerful and complex systems can be developed within small areas. In addition to the advantages brought by these integration technologies, new challenges are revealed. Increasing the number of devices in a given area often results in higher power density. This causes thermal problems which can severely degrade the reliability of modern electronic systems [30] [32] [33]. Low power architecture design and thermal management scheme are required. Also, more complex fabrication processes are required for modern integration technologies. How to improve the yield is a critical design issue for IC industry. In this dissertation, architecture designs and optimization technologies are proposed to improve power, reliability, and yield in cache, memory, and interconnect levels.

First, in cache level, a low power cache designed called *run-time reconfig-*

*urable expandable cache* is introduced. To obtain better power efficiency, L1 cache structures in most embedded systems are direct-mapped as opposed to set-associative where all lines in a set are activated during one access. Moreover, to further reduce power consumption, the size of L1 cache is limited. When applications run on embedded systems are simple and regular, direct-mapped cache is sufficient to provide acceptable behavior. However, when they are complex and irregular, direct-mapped cache tends to suffer from thrashing and cache pollution. For example, L. Lee, et al, [1] demonstrate the thrashing effect caused by MPEG4 codecs which are commonly included in modern hand-held devices. Note that the increase of L1 miss rate not only degrades the performance but also increase power consumption due to larger energy required to access data in L2. According to the experiment done by G. Bournoutian and A. Orailoglu [2], it may cost 20 times more energy to access data in L2 than it does in L1. Therefore, simple direct-mapped cache can no longer satisfy the requirements of modern embedded systems in terms of performance and power consumption. *Expandable cache* proposed by G. Bournoutian and A. Orailoglu [2] is efficient in reducing miss rate and energy consumption with small area overhead. However, the original *expandable cache* with only one expansion scheme may lead to thrashing problems. In this thesis, based on the structure of *expandable cache*, we will introduce a new cache design which has many expansion schemes to fit different run-time program behaviors. The expansion schemes of our proposed cache is dynamically changed by executing configuration instructions which are inserted at compile time.

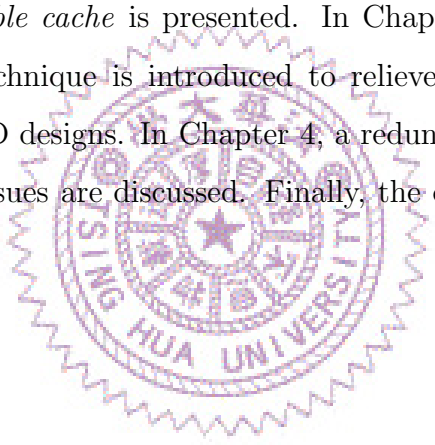
Second, a thermal-aware memory mapping technique for 3D designs is

proposed. With the capacity provided by modern integration technology, integrating memory chips into package has become popular in recent years. Several researches on memory integration based on SIP have been studied [25] [26] [27] [28] [29]. Though modern integration technology provides extremely high capacity for circuit integration, it suffers severe thermal stress because of three dimensional stacking of ICs [30]. Thermal stress will induce variation of DRAM retention time and reliability problem [31]. 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 logic block but also the physical location during memory mapping, second, the changing temperature of a physical block during execution of programs. In this thesis, we will propose a memory mapping algorithm taking into consideration the above-mentioned two points.

Finally, the recovery of failed TSV is discussed. TSV provides communication links for dies in vertical direction and is a critical design issue in 3D integration. Just like other components, the fabrication and bonding of TSVs can fail [47] [48]. A failed TSV can severely increase the cost and decrease the yield as the number of dies to be stacked increases. To improve the yield, some recovery mechanism for faulty TSV is needed. A simple but effective solution is to add redundant TSVs to replace failed TSVs. A re-

dundant TSV architecture with reasonable cost is proposed in this thesis. Our proposed redundant TSV scheme is scalable and can be adjusted to fit the failure rate of TSV for different TSV processes and bonding technologies. Given the failure rate of TSV and the number of TSVs required, probabilistic models are presented to compute the number of redundant TSVs that should be allocated so that an expected assembly yield can be achieved. Related design issues and design flow are discussed. The proposed redundant TSV design can successfully recover most of the failed chips and increase the yield to 99% based on probabilistic models.

The organization of this dissertation is as follows. In Chapter 2, to reduce the energy consumption in cache level, the structure of *run-time reconfigurable expandable cache* is presented. In Chapter 3, a thermal-aware memory mapping technique is introduced to relieve the thermal stress of memory system in 3D designs. In Chapter 4, a redundant TSV architecture and related design issues are discussed. Finally, the conclusion is presented in Chapter 5.



# Chapter 2

# Run-Time Reconfiguration of Expandable Cache for Embedded Systems

With the need to execute diverse and complex applications on embedded systems, the architecture of modern embedded systems is becoming increasingly complicated. Many hardware features such as multilevel caches that were originally designed for general purpose processors are now found in embedded processors. Although the capability of modern embedded processors has been greatly improved due to more sophisticated designs, the battery technology has been developed very slowly over the years. Therefore, the trade-off between power efficiency and performance is one of the most important issue to embedded systems.

To obtain better power efficiency, L1 cache structures in most embedded systems are direct-mapped as opposed to set-associative where all lines in a set are activated during one access. Moreover, to further reduce power consumption, the size of L1 cache is limited. When applications run on em-

bedded systems are simple and regular, direct-mapped cache is sufficient to provide acceptable behavior. However, when they are complex and irregular, direct-mapped cache tends to suffer from thrashing and cache pollution. For example, L. Lee, et al, [1] demonstrate the thrashing effect caused by MPEG4 codecs which are commonly included in modern hand-held devices. Note that the increase of L1 miss rate not only degrades the performance but also increase power consumption due to larger energy required to access data in L2. According to the experiment done by G. Bournoutian and A. Orailoglu [2], it may cost 20 times more energy to access data in L2 than it does in L1. Therefore, simple direct-mapped cache can no longer satisfy the requirements of modern embedded systems in terms of performance and power consumption.

Techniques have been proposed to improve direct-mapped L1 cache. These techniques can partitioned into two categories depending on whether additional level of memory is added. In the first category, a high-speed software-controlled on-chip SRAM called scratchpad memory (SPM) [3] is placed between processing element and L1 cache. Research has been conducted to copy highly used code segments, e.g., instructions in loop to SPM for low power designs [4] [5] [6]. Recently, the use of SPM has been extended to assign both program and data objects to SPM for energy reduction [7] [8]. Management issues related to the partitioning of SPM and data allocation have also been studied [9] [10]. Furthermore, the compiler-driven memory allocation schemes proposed by S. Udayakumaran et al. allows SPM to fully replace hardware cache [11]. SPM is effective in improving power efficiency for embedded systems. One feature of SPM is that the data objects to be

managed must be in a continuous data block. That is, SPM is good at handling data block in coarse granularity. However, when data to be accessed is spread out in a large memory space, using SPM can be very in-efficient.

The techniques in the second category focuses on cache design which allows the data to be processed in the granularity of cache set level. These techniques can be further classified to two types according to their objectives and motivations. The first type focuses on miss rate reduction or power efficiency. On-demand selective cache [12], dual data cache [13] and application-specific partitioned cache [14] are examples. In these techniques, the cache is divided into multiple partitions. Through run-time memory reference classification or compile-time program behavior analysis, data sets of different access behaviors can be stored in different cache partitions. Although these techniques lead to lower miss rate, they cannot be extended easily for complex, algorithmically intensive applications.

In the second type, techniques such as victim cache [15], pseudo-associative cache [16], and *expandable cache* [2] mimic the designs of set-associative cache to handle cache conflict problems by adding extra storage blocks or additional control logic. Among these techniques, *expandable cache* incurs relatively small area overhead and increases power-efficiency significantly. However, the original *expandable cache* with only one expansion scheme may lead to thrashing problems. In this work, based on the structure of *expandable cache*, we will introduce a new cache design which has many expansion schemes to fit different run-time program behaviors. The expansion scheme of our proposed cache is dynamically changed by executing configuration instructions which are inserted at compile time. The proposed method can be used in

high-volume embedded systems where the processor architectures are developed for specific applications and the instruction sets can be extended.

The rest of this chapter is organized as follows. Section 2.1 reviews the structure of *expandable cache* and gives the motivation of this work. In Section 2.2, our proposed cache design is presented. Next, the design flow and the required software tools are introduced in Section 2.3. The experimental results are presented in Section 2.4.

## 2.1 Observation & Motivation

In Section 2.1.1, *expandable cache* is introduced first. Then, in Section 2.1.2, we point out that *expandable cache* is not optimal by examples and discuss the properties that a newly designed cache should have in order to fully make use of the concept of *expandable cache*.

### 2.1.1 Expandable Cache

For a direct-mapped cache, when multiple data blocks with identical set indices are accessed in the same period, only one of them can exist in the cache. This results in conflict misses. To prevent cache conflict problem described above, some modification must be made to allow more than one data blocks with the same set indices to be stored in a cache. *Expandable cache* shown in Figure 2.1 proposed in [2] allows the expansion of each cache set into a secondary cache set. Therefore, two data blocks with the same set indices can be stored in the cache simultaneously. The circular list shown at the lower-right of Figure 2.1 records the indices of the sets which are most recently evicted. When a cache miss occurs and the index of the accessed set

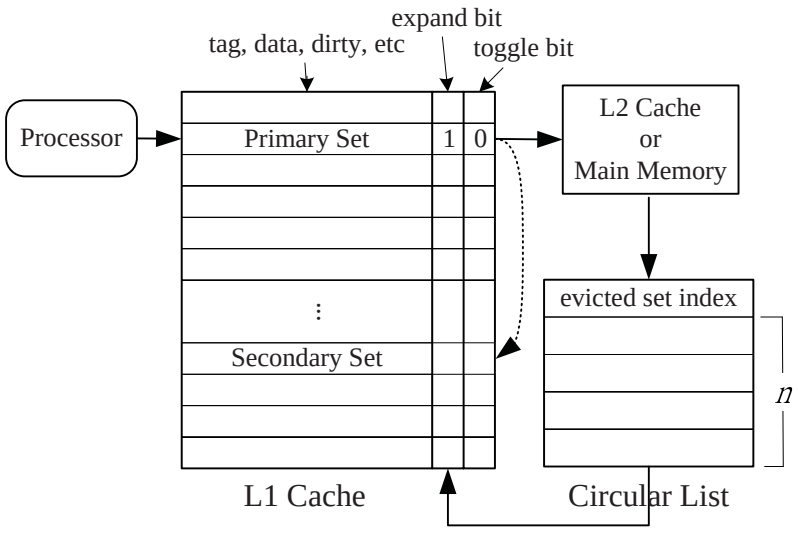



Figure 2.1: Dynamically Expandable L1 Cache

is already presented in the circular list, it implies that the current set is in a probable state of thrashing and should be expanded. To expand a set, the *expand bit* of the set is turned on. The index of secondary set is obtained by flipping the Most Significant Bit (MSB) of primary index.

During the execution, on a cache miss, if the *expand bit* is enabled, a secondary set is accessed on the next cycle. The *toggle bit* of a set is disabled whenever a hit occurs on the primary set and enabled whenever a hit occurs on the secondary set. This encodes Most Recently Used (MRU) scheme and determines whether primary or secondary set should be accessed first. If both primary and secondary sets result in cache misses, then one of these two sets needs to be selected as the evicted set and moved out from the cache. The *toggle bit* also determines which set should be evicted on a cache miss.

By adding minimal amount of extra storage, *expandable cache* allows each access to re-lookup into a predefined secondary set. This mechanism doubles

the size of a set without doubling the hardware. The major concern of this cache structure is whether the expansion of a set may cause a conflict problem of the secondary set which contains some data in use. In [2], the secondary set is chosen by flipping the MSB of the set index and hence is located at the farthest distance from the primary set. This solution is an intuitive choice based on the principle of locality, i.e., the data in the farthest distance may not be used immediately. However, due to the complexity and the irregularity of modern embedded applications, this expansion may not always be optimal. Furthermore, the lack of ability to prevent a primary set to be expanded to the secondary set that contains frequently used data may result in a higher miss rate as compared with the non-expandable cache.

### 2.1.2 Thrashing of Expandable Cache

In this section, a simplified example is given to show that using MSB for cache expansion proposed in [2] may not be optimal. The code given in Figure 2.2(a) contains six arrays which map to a direct-mapped cache with 8 sets. The mapping relation between arrays and sets is shown in Figure 2.2(b) and (c). For a given $i$ range, the sets in white background are active sets while those in gray background are currently unused sets. The lines with arrow depict the expanding relations of each primary and secondary set pair where tail represents primary set and heads its secondary set. When MSB is used for expansion and $i$ is between 0 to 7, the left part of Figure 2.2(b) illustrates the conflicting situation. Since both arrays $A$ and $E$ are mapped to set 000, the set is expanded to set 100 by flipping MSB. However, set 100 contains the data of array $C$ which is also in use. Therefore, the thrashing

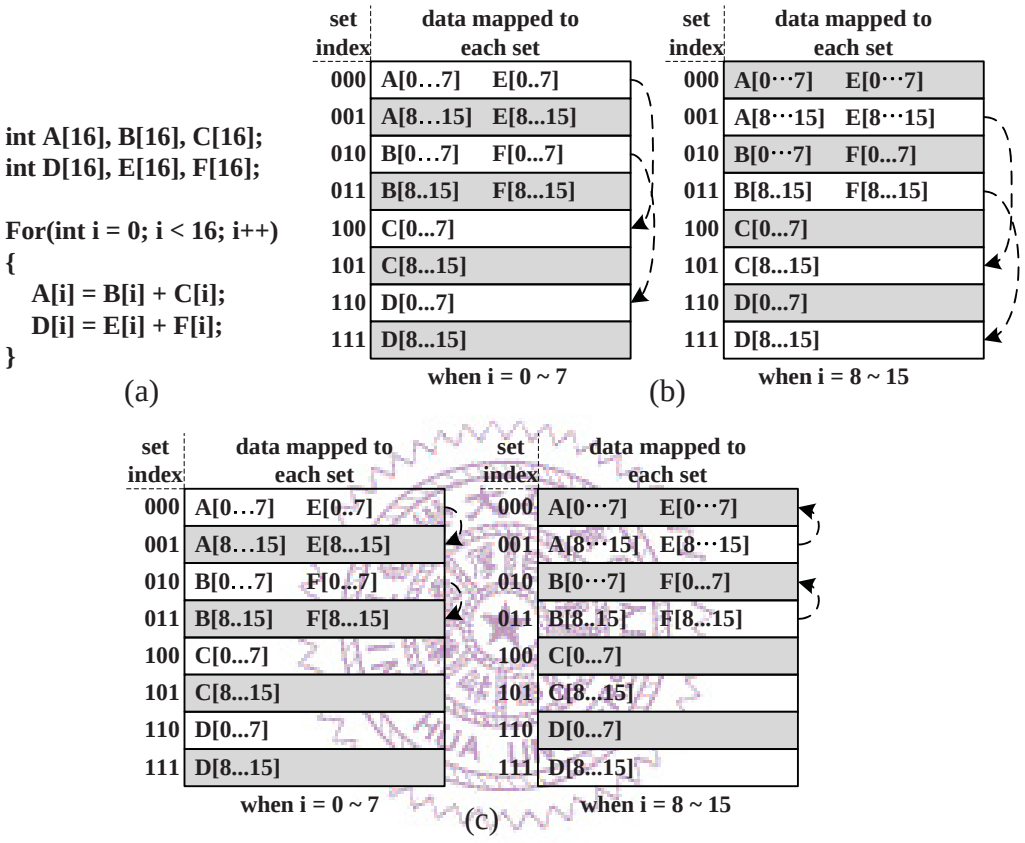



Figure 2.2: (a) Example of Trashing Code; (b) Expansion based on MSB; (c) Expansion based on LSB

problem occurs. The same situation happens with sets 010 and 110. Even when $i$ changes its value to the range of 8 to 15, the problems are equally serious as shown on the right part of Figure 2.2(b). In this example, we found that though half of the sets are inactive, using MSB for expansion causes serious conflict problem. On the contrary, using Least Significant Bit (LSB) for expansion provides a very good expansion scheme by fully utilizing inactive sets as shown in Figure 2.2(c). Nevertheless, expansion using LSB may not be a good choice for other applications. For different applications or program segments, the optimal expanding schemes can be very different. Therefore, we propose in this chapter to have dynamic expansion scheme that is configurable during the execution of different applications or program segments.

In fact, the complexity and the irregularity of modern embedded applications highly disturb the locality of data access. For example, instead of accessing an array of integers sequentially, it is common for a program segment to traverse an array of large structures where only a fixed data field of each structure needs to be accessed. This situation can also be found in multi-dimensional arrays, where access is not based on the index of the last dimension. Moreover, a set containing variables which are accessed with extremely high frequencies in a program should never be allowed to be used as a secondary set. To fully utilize the concept of *expandable cache*, a more flexible expanding mechanism is required. **In the best case, we should always select inactive sets as secondary sets.** To these ends, the desired properties of an expandable cache are listed as follows:

- The cache should have the ability to prevent any frequently accessed

set from becoming a secondary set.

- The expansion scheme is configurable to fit different program segments dynamically.
- The hardware for expansion should be simple and flexible enough to execute a number of expansion schemes.

## 2.2 Run-Time Reconfigurable Expandable Cache

Over a short execution interval, the percentage of sets that contain data to be reused recently in a cache is often less than 50%. This means that in normal situation, there should be sufficient unused sets to be used for set expansion. Based on the motivation described in Section 2.1.2, the optimal expansion scheme is to expand an active set to an unused secondary set. However, this simple strategy is impossible to be carried out due to the unpredictable irregular distribution of active and inactive sets in a cache. Even if the distribution of active and inactive sets can be predicted, the hardware to execute different expansion schemes may be too expensive. Furthermore, the desired expansion scheme may need to change very often to fit different applications or execution stages. Thus, we consider only expansion schemes that incur small area and power overhead.

The limitation of *expandable cache* [2] is that only MSB is used to determine the secondary set. Hence, the first improvement to *expandable cache* is to allow any bit of the set index to be flipped to obtain the secondary set. Next, to cope with the increasing complexity and irregularity of modern embedded software, more than one bit is allowed to be flipped. Based on the

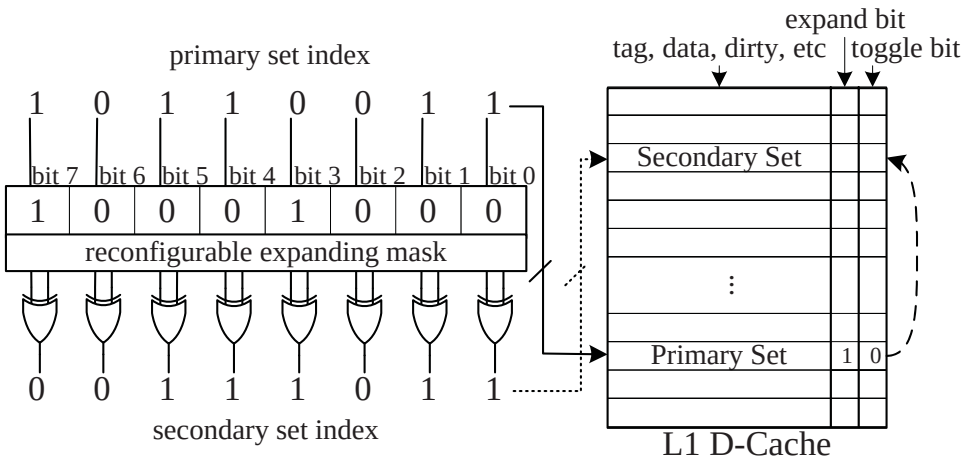



Figure 2.3: Expansion Hardware

discussion above, the expansion hardware shown in Figure 2.3 is proposed for our cache design. In this expansion hardware, we have a register to store the *reconfigurable expanding mask* which is used to map a primary set index to its secondary set index by XOR operations. In this design, only a register and a small number of XOR gates are added. For example, when the value of *reconfigurable expanding mask* is equal to 10001000 where *bit 7* and *bit 3* are set to 1, the index of secondary set is obtained by flipping *bit 7* and *bit 3* of the primary set index. Thus, given a primary set, *set 10110011*, the secondary set is computed as *set 00111011*. The cache structure on the right part of Figure 2.3 is similar to that of *expandable cache*. Both *expand bit* and *toggle bit* in the original design [2] are preserved in our cache design. The *expand bit* of a set indicates whether the set is expanded and the *toggle bit* which encodes an MRU scheme determines which set should be accessed first and which set should be evicted on a cache miss.

The circular list in the original *expandable cache* [2] which is used for

automatic set expansion is removed since it is no longer sufficient to handle the complex expansion problem. Instead, a number of new instructions used for set expansion are defined as follows for the new architecture.

- **set_expand_bit** $\boldsymbol{X}$;

  Enable the *expand bit* of set $\boldsymbol{X}$.

- **clear_expand_bit** $\boldsymbol{X}$;

  Disable the *expand bit* of set $\boldsymbol{X}$.

- **set_expand_mask** $\boldsymbol{M}$;

  Load *reconfigurable expanding mask* with $\boldsymbol{M}$.

- **clear_all_expand_bits**;

  Disable the *expand bits* of all sets in the cache.

Through insertion of these instructions in a program, the expanding and recovering of each set can be controlled at compile time and therefore becomes more flexible. By loading *reconfigurable expanding mask* with different values, various expansion schemes can be implemented.

To reduce inserted instruction count, instructions ***set_expand_bit X*** and ***clear_expand_bit*** are extended to allow *expand bits* of multiple sets to be enabled or disabled in one instruction. Two more instructions are defined as follows.

- **c_set_expand_bit** $\boldsymbol{X, d, n}$;

  This instruction takes $\boldsymbol{n + 1}$ cycles for execution. A register, $\boldsymbol{s}$, is required to store the index of the set to be expanded. In the first cycle, the value of $\boldsymbol{X}$ is loaded into register $\boldsymbol{s}$. In the following $\boldsymbol{n}$ cycles,

The value of ***s*** is increased by ***d***. For example, executing instruction "***c_set_expand_bit 101, 8, 2***" causes the *expand bits* of sets 101, 109, and 117 to be enabled.

- **c_clear_expand_bit *X, d, n***;

  The operation of this instruction is similar to instruction ***c_set_expand_bit X, d, n***. Instead of enabling the *expand bits* of sets, this instruction disables *expand bits*.

The two instructions introduced above are efficient in reducing the static instruction count which is critical for embedded systems. However, the number of cycles required to enable or disable sets remain unchanged.

Now we discuss the overhead incurred by our scheme. The first one is power consumption. The flip-flops in Figure 2.3 are set when instruction ***set_expand_mask M*** is executed. Power consumption of these instructions is increased only when flip-flops are written. According to experiments, the execution frequency of this instruction is less than 0.1%. The power impact is low. In normal execution mode, flip-flops remain constant and only power of XOR gates are incurred. The gate count of a modern embedded processor such as ARM926EJ-S is around 0.16M $\sim$ 0.18M where the power consumption is around 0.235mW/MHz under 90nm technology [17]. The worst case power consumption of an XOR gate under the same process technology is around 30nW (including static and dynamic power). The power consumption of these XOR gates is insignificant. As for timing issue, our approach will increase the critical path delay for at most one XOR gate delay. However, in modern pipeline processors, the performance bottleneck is often found in the

stage of register file access due to the increasing number of registers and the requirement of multi-port access [18] [19]. Our proposed expansion hardware is added to L1 data cache in the stage of memory access. Therefore, the overall system performance will not degrade. Lastly, since all operations for cache expansion are done through instruction level commands, there will be extra number of static and dynamic instruction counts added to programs.

## 2.3 Software Design Flow

In this section, the overall design flow is presented in Section 2.3.1. Then, the software tools in our design flow are introduced in Section2.3.2 with details. Before our presentation, following terms are defined.

***code segment:*** A *code segment* is a sequence of static code instructions in a loop or a function. Multiple *code segments* can be nested with one another.

***interval:*** An *interval* refers to a period time that the instructions of a *code segment* are executed. For a given *code segment*, multiple *intervals* can be defined based on the execution flow of a program.

***expansion scheme:*** For a given *interval*, the values of expanding mask and the expand bits of all cache sets are referred as the *expansion scheme* of that *interval*.

***coverage ratio:*** To setup the values of expanding mask and the expand bit of each cache set, configuration instructions need to be executed. These

configuration instructions lead to overhead in terms of static and dynamic instruction counts. The overhead of selecting *expansion schemes* of all *intervals* can be very large. Therefore, for given a *code segment* and all its *intervals*, only a limited number of *expansion schemes* are selected. An *interval* to have its own *expansion scheme* selected is called a *covered interval*. For a given *code segment*, the term *coverage ratio* is defined as the execution cycle count of the *covered intervals* divided by the execution cycle count of all *intervals*. We expect the value of *coverage ratio* of *code segment* to be larger than a used defined *coverage ratio*.

### 2.3.1 Design Flow

In our proposed architecture, configuration instructions are inserted in compile time. An example is depicted in Figure 2.4. The code structure of the original program is shown in the left part of Figure 2.4. According to the execution flow, the program is partitioned into three *code segments*, *code segments A*, *B*, and *C*. For different *code segments*, different program behaviors are observed and different *expansion schemes* of *expandable cache* are required. To allow different *expansion schemes* to be configured, configuration instructions are inserted prior to each *code segments*, as shown in the right part of Figure 2.4. For programs with complicated behaviors, a *code segment* may require different *expansion schemes* over different *intervals*. For example, *code segment C* in Figure 2.4 requires different *expansion schemes* for $i = 1 \sim 5000$ and $i = 5001 \sim 10000$, respectively. In this case, the *code segment* is duplicated and different configuration instructions are inserted.

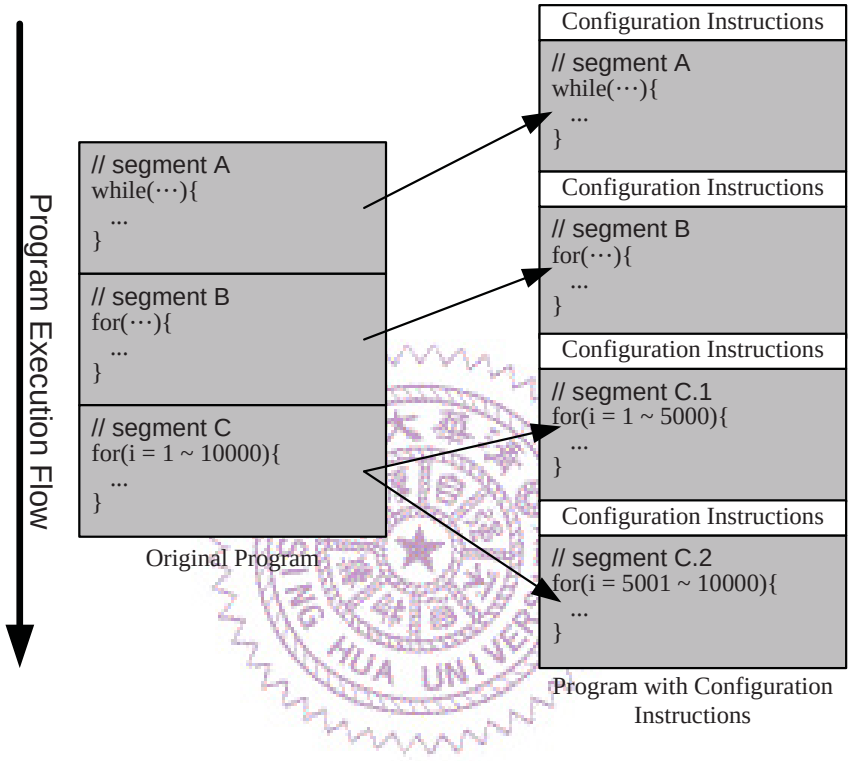



Figure 2.4: Insertion Scheme of Configuration Instructions

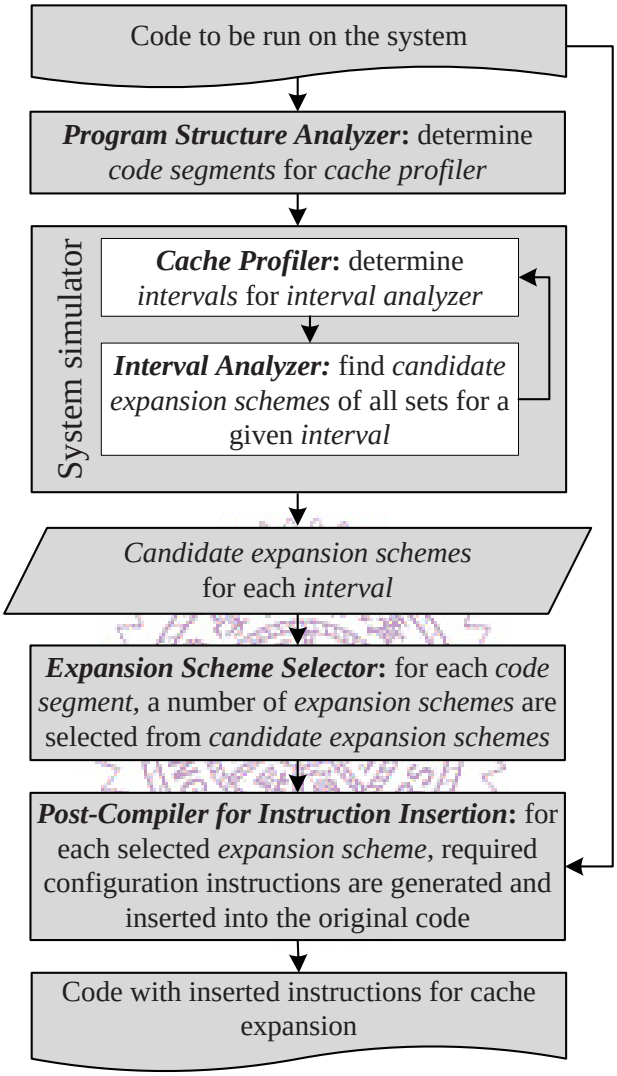



Figure 2.5: Design Flow

To realize the proposed instruction insertion scheme, four steps are designed at software level and listed as follows:

- **Determine *code segments***

  The configuration instructions for cache expansion cannot be inserted in a program execution flow arbitrarily. Program structure and execution flow need to be considered. *Program structure analyzer* is developed to find candidate positions such as entry points to loops and functions for instruction insertion. Bodies of loops and functions are identified as possible *code segments* for cache expansion.

- **Determine the *intervals* for different *expansion schemes* in a *code segment***

  For each *code segment*, different *expansion schemes* may be required for different *intervals* or different execution stages. *Cache profiler* is designed to determine *intervals* during the execution of *code segments*. The sizes of the *intervals* for a *code segment* need not to be identical. Based on the profiling data collected by *cache profiler* during simulation, *intervals* of different sizes are allowed. The details of determining an *interval* is explained in Section 2.3.2. Whenever an *interval* is found, *interval analyzer* is invoked.

- **Find the candidate *expansion schemes* for a given *interval***

  For each *interval* determined by *cache profiler*, a number of *candidate expansion schemes* leading to minimal cache misses are determined by *interval analyzer*. *Cache profiler* and *interval analyzer* are invoked iteratively till the simulation of the whole program is completed.

- **Select *expansion schemes* and insert configuration instructions**

  For each *code segments*, *expansion scheme selector* is developed to select a number *expansion schemes* from the *candidate expansion schemes* in order to reduce the amount of inserted instructions and duplicated codes. Next, the configuration instructions for the selected *expansion schemes* are generated and inserted into the original code by a *post-compiler*.

The overall design flow is depicted in Figure 2.5. *Program structure analyzer*, *expansion scheme selector*, and *post-compiler for instruction insertion* are implemented as program modules and are invoked by user individually. *Cache profiler* and *interval analyzer* are implemented inside the system simulator and are invoked automatically during the simulation. The dedicated description is presented in Section 2.3.2.

### 2.3.2 Software Tools in the Design Flow

This section introduces the software tools required for our design flow. First, *program structure analyzer* is introduced in Section 2.3.2. *Cache profiler* is introduced in Section 2.3.2. The algorithm for *interval analyzer* is explained in Section 2.3.2. *Expansion scheme selector* is presented in Section 2.3.2. Configuration instruction insertion for cache expansions is described in Section 2.3.2. Finally, the complexity and the efficiency of the proposed software flow is discussed in Section 2.3.2.

### Program Structure Analyzer

*Program structure analyzer* is developed to analyze the execution flow of each program. Candidate positions for instruction insertion are the entry points to loops and functions. These positions are identified by analyzing branch instructions. According to the type and target address of each branch instruction, *code segments* of loops and functions are determined. Note that different *code segments* can be nested to each other. The analysis in this stage is static.

### Cache Profiler

*Cache profiler* is embedded in a system simulator. The main objective of *cache profiler* is to find *intervals* which need new cache *expansion schemes* inside each *code segment*. During the simulation, whenever a *code segment* is entered, the *cache profiler* is invoked to initiate a new set of profiling data. While the simulator leaves the *code segment*, the collection of profiling data for this *code segment* is ended. Multiple sets of profiles may be maintained at the same time due to the nested structure of *code segments*. Run-time information such as execution count and execution duration of each *code segment* are recorded. This information is required in *expansion scheme* selection and configuration instruction insertion stages.

Before we start to design the procedure, it is necessary to understand when a set is worth expanding. That is, when expanding a set into its secondary set, the total miss count should be truly reduced. This problem can be discussed from two aspects. First, we need to precisely know whether a set suffers cache conflict problem. Second, we need to know whether a

secondary set can be found. For a given *interval*, both these two problems can be answered according to the values of *miss count* and *reference count*. If a set suffers from thrashing, the *miss count* must be relatively large. As for the second problem, a set with high *reference count* is not a suitable secondary set. Making such a set a secondary set may induce a large number of additional cache misses. Thus, from the above observations, it is preferable to expand a set with high *miss count* to a secondary set with small *reference count*.

As to *interval* determination, when the number of sets with high *miss count* is no greater than the number of sets with low *reference count*, it can be considered a suitable *interval* for cache expansion since there are sufficient sets to become secondary sets. If the *interval* is short, this condition can be easily satisfied. However, considering the overhead of extra instructions inserted for each *interval*, the *interval* should be as large as possible. Nevertheless, as the *interval* becomes larger, the number of sets with high *miss count* increases and the number of sets with low *reference count* decreases.

Hence, in our algorithm, the following steps are performed to decide whether an *interval* is reached. First, *average-miss-count* and *average-reference-count* are defined as the averages of *miss counts* and *reference count* of all sets, respectively. Second, *miss count* of a set greater than *average-miss-count* is defined as *high-miss-count* and otherwise as *low-miss-count*. Similarly, *high-reference-count* and *low-reference-count* are defined. Then, two thresholds, $upper\_bound_{\#high\_miss\_sets}$ and $lower\_bound_{\#low\_ref\_sets}$, are given as the upper bound of the number of sets with *high-miss-count* and the lower

bound of the number of sets with *low-reference-count*. Whenever the number of sets with *high-miss-count* becomes larger than $upper\_bound_{\#high\_miss\_sets}$ or the number of sets with *low-reference-count* becomes smaller than $lower\_bound_{\#low\_ref\_sets}$, the *interval* should not be further extended.

As for selecting the values of $upper\_bound_{\#high\_miss\_sets}$ and $lower\_bound_{\#low\_ref\_sets}$, the following experiment is performed. In the ideal case, the number of sets with *high-miss-count* is equal to the number of sets with *low-reference-count* and each set with *high-miss-count* is expanded to a set with *low-reference-count*. In this case, the numbers of sets with *high-miss-count* and that with *low-reference-count* are equal to 50% of the total number of sets. The value of $lower\_bound_{\#low\_ref\_sets}$ is set using the above mentioned analysis. Then, with the value of $lower\_bound_{\#low\_ref\_sets}$ set to 50% of the total number of sets, experiments are conducted for different values of $upper\_bound_{\#high\_miss\_sets}$. Thus, the value of $upper\_bound_{\#high\_miss\_sets}$ starts from 50% and decreases at intervals of 5% for each trial. Based on experimental results, setting the value of $upper\_bound_{\#high\_miss\_sets}$ to 30% of the total number of sets is beneficial for most test cases. These values are fixed in all experiments in Section 2.4.

The algorithm implemented in *cache profiler* is depicted in Figure 2.6 and described as follows. In order to run *cache profiler* robustly, we define *minimum interval* which is the shortest *interval* to be analyzed for an *expansion scheme* and *base interval* which is the size of an *interval* to be extended and checked periodically. *Minimum interval* is to guarantee that the change of *expansion scheme* is at a frequency less than $\frac{1}{minimum\ interval}$ and thus limits the number of added instructions. During simulation, *cache profiler* keeps

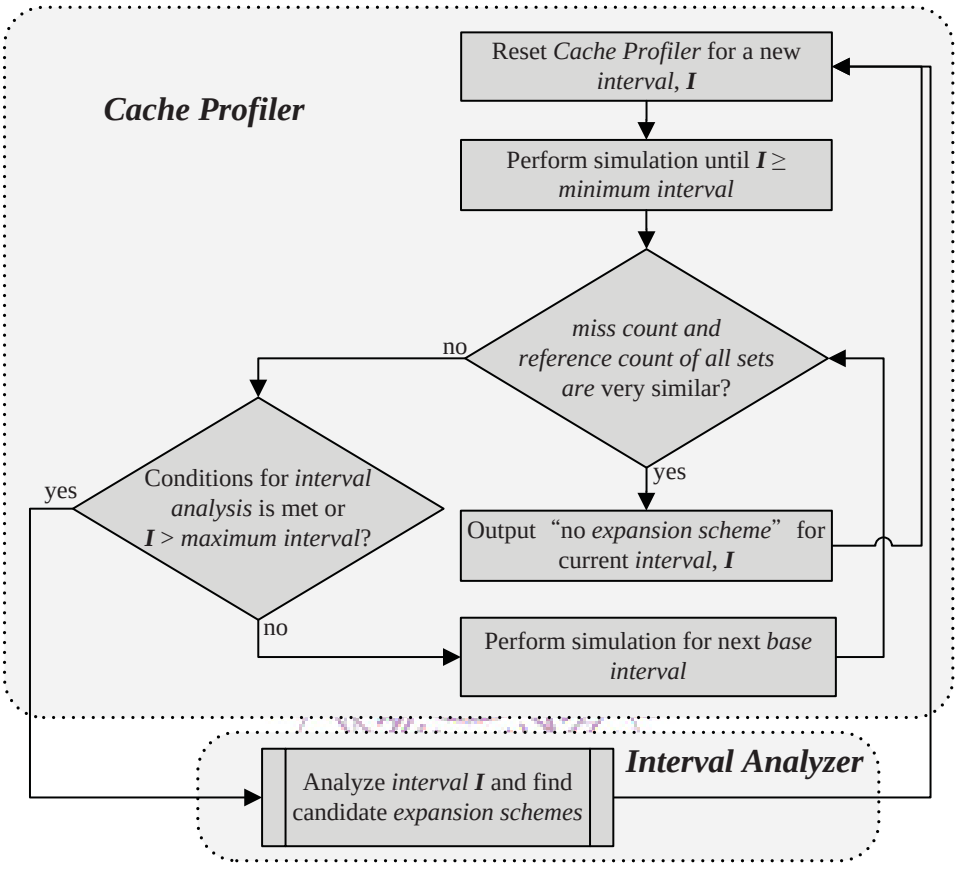



Figure 2.6: Flow Chart of Software Tools in System Simulator

track the *reference count* and *miss count* of each set. First, when a simulation range is greater than or equal to *minimum interval*, the profiler calls an inspector to inspect collected data. If *miss count* and *reference count* of all sets are very similar, the profiler determines that no expansion to be made for current *interval* and starts a new iteration of profile. Otherwise, the profiler checks whether the number of sets with *high-miss-count* is larger than $upper\_bound_{\#high\_miss\_sets}$ or the number of sets with *low-reference-count* is smaller than $lower\_bound_{\#low\_ref\_sets}$. If any of these two conditions is true, *cache profiler* stops extending current *interval* and *interval analyzer* (described in Section 2.3.2) is invoked to determine *expansion schemes*. To prevent the *interval* from growing too large, *maximum interval* is also defined. If *maximum interval* is reached, *interval analyzer* is invoked, too. Otherwise, simulation for the next *base interval* is preformed to extend the current *interval*.

**Interval Analyzer**

After each *interval* is recognized, *interval analyzer* is invoked to determine the *candidate expansion schemes*. In Section 2.3.2, we pointed out that it is preferable to expand a set with *high-miss-count* to a secondary set with *low-reference-count*. We would like to compute the *gain* of different kinds of expansions. For each *reconfigurable expanding mask*, we define its gain function, $MaskGain$. Before we define $MaskGain$ of a *reconfigurable expanding mask*, we define the gain function of a set, $SetGain$, in this configuration. For a set, since the expected miss reduction is bounded by *miss count* of primary set, $P$, and *reference count* of its secondary set, $S$, we define the

gain of a set as

$$SetGain(P) = miss\ count(P) - reference\ count(S).$$

If a *SetGain* becomes negative, we set it to zero. This implies that, if the configuration is adopted, the expand bit of set $P$ is disabled and the cache miss reduction of this set is zero. Then, for a given value, $V$, of *reconfigurable expanding mask*, $MaskGain$ is computed as

$$MaskGain(V) = \sum_{P=0}^{2^n-1} SetGain(P),$$

where $n$ is the bit-length of set index. The *interval analyzer* computes the $MaskGain$s of all possible values of *reconfigurable expanding mask* and records the best 20% of them. In later *expansion scheme* selection stage, the exact *expansion schemes* for a given *code segment* are selected from these *candidate expansion schemes* taking into consideration the increase in code size.



**Expansion Scheme Selector**

For each static *code segment* determined by *program structure analyzer* introduced in Section 2.3.2, one or more *intervals* are analyzed for candidate *expansion schemes* by *interval analyzer* introduced in Section 2.3.2. Theoretically, it is possible to configure the best expansion masks for all *intervals* through compiler techniques such as loop unrolling and code duplication. However, this may result in undesired growth of code size. Therefore, some policy is required to determine whether compiler techniques should be applied to insert configuration instructions for different *intervals* of a *code segment*.

Two selection algorithms are proposed to select expansion configurations in a static *code segment*. The first technique is proposed to handle nested code structures such as nested loops. Take a 2-level nested loop as an example. For each *interval* determined by *cache profiler* introduced in Section 2.3.2, the number of dynamic instruction count is recorded to estimate the execution time of that *interval*. Let the execution time of the inner and outer loops be denoted as $T_{inner}$ and $T_{outer}$, respectively. If $(T_{inner})/(T_{outer} - T_{inner})$ is greater than a predefined threshold value, $\tau$, the *expansion schemes* of the outer loop are discarded. In this case, the configuration instructions required by the inner loop can be moved to the entry point of the outer loop to reduce the overhead of dynamic instruction count. Otherwise, the *expansion scheme* of the outer loop is taken and the *expansion scheme* of the inner loop is discarded. In our experiments, $\tau$ is set to 100.

The second technique is used when different *expansion schemes* are required for one static *code segment* over different *intervals*. For this kind of execution flow, the algorithm presented in Figure 2.7 is developed for *expansion scheme* selection. The input of the algorithm contains a set of *intervals* which are required for one static *code segment*. For each *interval*, a set of candidate *expansion schemes* are recorded in the step of *interval analysis*. The output of the algorithm is a set of selected candidate *expansion schemes*. The detailed execution flow of the proposed algorithm is explained as follows.

Set $\boldsymbol{I}$ contains all *intervals* of a static *code segment*. Set ***all_scheme*** is the candidate *expansion schemes* of all *intervals* in set $\boldsymbol{I}$. Let $\gamma$ stand for the used defined *coverage ratio* and $n$ for the maximum number of selected *expansion schemes*. Figure 2.7 shows the algorithm. Let $\boldsymbol{S}$ stand for the

set of *expansion schemes* that are selected. At first, set $\boldsymbol{S}$ is initialized to an empty set (Line 1). In each iteration of the while loop (Line 3-14), for each *expansion scheme*, the summation of the gain function developed in Section 2.3.2, *mask_gain*(), for all *intervals* in $\boldsymbol{I}$ are computed (Line 5-6). The best *expansion scheme* is selected (Line 7). Then, all *intervals* that are covered by the selected *expansion scheme* are removed from set $\boldsymbol{I}$ (Line 9-11). The while loop stops when the *coverage ratio* is larger than $\gamma$ or the number of selected *expansion schemes* reaches $n$. In general, a larger $\gamma$ is beneficial for cache miss reduction. However, it results in large number of *expansion schemes* and increases the number of static instruction count. Experiments are conducted to select the value of $\gamma$. We found that when *coverage ratio* is larger than 90% or the number of selected *expansion schemes* is larger than 8, the increase of *coverage ratio* due to a newly included *expansion scheme* becomes very small ($< 3\%$). Therefore, in our experiments, $\gamma$ and $n$ are set to 90% and 8, respectively.

**Configuration Instruction Insertion**

After the exact *expansion scheme* of each *interval* is determined, the next step is to determine the configuration instructions to be inserted. For a given *interval* and its *expansion scheme*, the instruction "***set_expand_mask M***" is always required unless the value of $\boldsymbol{M}$ for the current *interval* is identical to that of the previous *interval*. Next, instructions to configure the *expand bit* of each set are inserted. To achieve the desired configuration, we can either use "***clear_all_expand_bits***" instruction to disable all bits and then use "***set_expand_bit X***" instruction to enable bits, or we can

**Algorithm: program *Expansion Scheme Selection*()**

**Inputs:**

| | |
|---|---|
| $\boldsymbol{I}$ | : A set of *intervals* for a given *code segment*. |
| ***all_schemes*** | : All candidate *expansion schemes* of the *intervals* in $\boldsymbol{I}$. |
| $\gamma$ | : The used defined *coverage ratio*. |
| $n$ | : The maximum number of selected *expansion schemes*. |

**Output:**

| | |
|---|---|
| $\boldsymbol{S}$ | : A set of selected expansion schemes. |

```
1  S = φ
2
3  while(current coverage ratio < γ AND #selected expansion schemes ≤ n)
4  {
5    for each expansion scheme c in all_schemes, do
6      total_mask_gain(c) = the mask_gain(c) for all intervals in I;
7    best_scheme = the expansion scheme with best total_mask_gain(c);
8
9    for each interval i in I, do
10     if(best_scheme is a candidate expansion scheme of i)
11       remove i from I;
12
13   add best_scheme to S;
14 }
15
16 return S;
```

Figure 2.7: Algorithm for *Expansion Scheme Selection*

use "***set_expand_bit X***" and "***clear_expand_bit X***" instructions to modify the configuration of the previous *interval*. The instruction count for the first method is equal to 1 plus the number of bits to be enabled. The instruction count for the second method is equal to the number of bits to be modified. The one with fewer instruction count is selected as our solution. Then, extended instructions, "***c_set_expand_bit X, d, n***" and "***c_clear_expand_bit X, d, n***", are used to reduce static instruction count by replacing multiple "***set_expand_bit X***" or "***clear_expand_bit X***" instructions.

### Complexity and Efficiency of the Proposed Software Design Flow

*SimpleScalar* [20] is used as our system simulator and 17 programs from *CPU SPEC2000* [21] benchmark suite are selected as test cases. For nine test cases, the time required for system simulation is less than 4 hours on a desktop with a 3.2GHz dual-core processor. For seven test cases, the simulation time ranges from 6 to 12 hours. One test case, `188.ammp`, requires 15 hours for the simulation to complete. System simulation is the bottleneck in terms of run time. In our proposed software design flow, *cache profiler* and *interval analyzer* are incorporated into a system simulator and invoked iteratively. The increase of simulation time due to *cache profiler* and *interval analyzer* is analyzed as follows.

During the simulation, *cache profiler* needs to maintain the values of *average-reference-count* and *average-miss-count* so that the values of $\#low\_ref\_sets$ and $\#high\_miss\_sets$ can be determined. For each set, two counters are maintained to provide *miss-count* and *reference-count* locally.

One *average-reference-count* counter and one *average-miss-count* counter are also maintained globally. For each reference to cache, these counters are updated.

Next, the complexity of *interval analyzer* is discussed. For a cache with $n$-bit set index, the number of sets in the cache is $2^n$. For a given value of *reconfigurable expanding mask*, $2^n$ substraction and branch operations are required to compute the $SetGain$s of all sets. Then, $(2^n - 1)$ addition operations are performed to compute $MaskGain(V)$. Since the the number of possible values of *reconfigurable expanding mask* is $2^n - 1$, the complexity can be denoted as

$$O(((2 \cdot 2^n) + (2^n - 1)) \cdot (2^n - 1)) = O(2^{2n}).$$

Although the complexity of *interval analyzer* is exponential, the value of $n$ is usually no greater than 10 in L1 cache of a modern embedded processor. When $n = 8$, one run of *interval analyzer* takes less than $1ms$.

Experiments are conducted to observe the increase of simulation time when *cache profiler* and *interval analyzer* are incorporated into the system simulator where $n$ is set to 8. When only *cache profiler* is incorporated into the system simulator, the simulation time is increased by 0.17% in average. When both *cache profiler* and *interval analyzer* are incorporated, the simulation time is increased by 6.84% in average. As compared with the time required for system simulation, the time require by the *cache profiler* and *interval analyzer* is insignificant.

## 2.4 Experimental Results

In this section, experiments are conducted to evaluate the effectiveness of our proposed cache structure and software design flow. In Section 2.4.1, the setup of our experiments is explained. In Section 2.4.2, experimental results for test cases selected from *SPEC CPU2000* [21] are reported. Finally, in Section 2.4.3, experiments are conducted to observe the miss rate of the proposed cache structure when different input files are fed to each test case.

### 2.4.1 Experiment Setup

The cache structure presented in this work is proposed to allow complicated applications to be executed efficiently on embedded systems. Test cases with large input data and complicated behaviors are desired. For this purpose, test cases from *SPEC CPU2000* [21] benchmark suite are selected. *SimpleScalar* toolset [20] incorporated with *cache profiler* and *interval analyzer* is used for system simulation. A 32-bit processor with in-order, single-issue, and separate instruction and data caches is assumed.

In the original *expandable cache* [2], *expand bits* need to be cleared to flush or reset the cache. Without cache flush or reset mechanism, *expand bits* of all cache entries will be enabled all the way to the end. Clearing *expand bits* (cache flush or reset) is an important step in the original *expandable cache* to determine the performance. However, the conditions of cache flush or reset are not defined in the previous work [2]. We could not exactly reproduce the experimental results [2]. Therefore, we take a more rigorous comparison: miss rate of 2-way set-associative cache and the energy consumption of the original

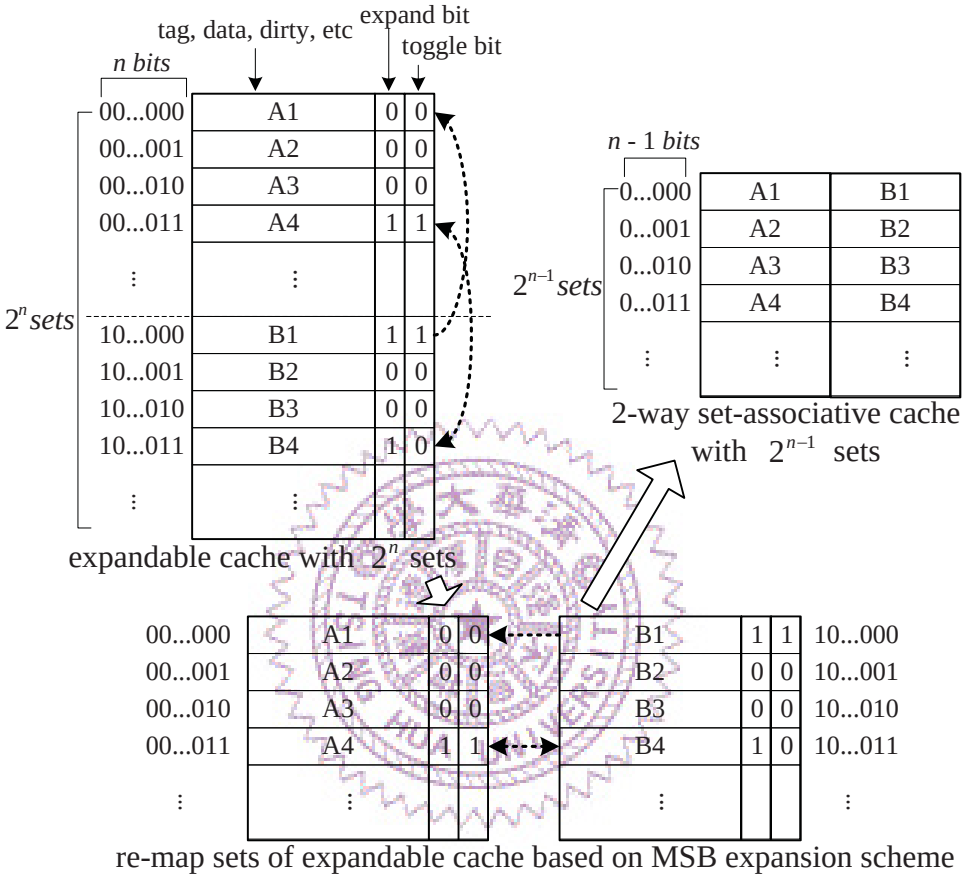



Figure 2.8: Reduce Expandable Cache to 2-Way Set-Associative Cache

Table 2.1: L1 Data Cache Structures for Experiments

| **Cache Parameters** | **2-Way Set-Associative Cache (≈Expandable Cache)** | **Our Proposed Cache Structure** |
|---|---|---|
| Cache Size | 8 KB | 8 KB |
| Line Size | 32-byte | 32-byte |
| Set Number | 128 | 256 |

*expandable cache* are assumed. The miss rate of the original *expandable cache* [2] can never be better than that of a 2-way set-associative cache where the number of sets is halved. This can be explained by Figure 2.8. Since MSB is used to determine the secondary set, we can view the lower half sets of the original direct-mapped cache as the second-way of a 2-way set-associative cache. The major difference is that instead of accessing two sets associatively, *expandable cache* accesses these sets one at a time based on the structure of direct-mapped cache. In terms of power efficiency, *expandable cache* is superior because only one set is accessed each time. The miss rate of 2-way set-associative cache is the lower bound of that of *expandable cache*. Therefore, we take the miss rate of 2-way set-associative cache to represent that of *expandable cache* for comparison. The L1 cache structures used in our experiments are summarized in Table 2.1. With 8 KB cache size and 32-byte line size, our proposed cache have 256 sets while 2-way set-associative cache has 128 sets.

In addition to miss rate reduction, the energy usage improvements on data cache is also reported. Since the energy usage improvement of this work is based on L1 cache miss reduction, we need to define the structure of L2 cache so that the energy ratio of accessing data in L1 to L2 can be known.

Similar to [2], L2 was assumed to be 256 KB with 4-way set-associative and 64-byte line size. The feature size of the hardware is assumed to be 65nm. By using *CACTI* [22], the energy required to access data in L2 is 20 times more than that to L1. Based on this ratio, the energy consumed can be defined as

$$E_{cache} = E_{unit} \times (AccessCount + \delta) + 20 \cdot E_{unit} \times MissCount_{L1}$$

where $E_{unit}$ represents the energy required to access data in L1 and $\delta$ represents the extra access count for expanded sets. According to our experiments, the value of $\delta$ is found to be between 3%∼8% of the total data cache access count for all test cases.

### 2.4.2 Experimental Results for Miss Rate and Energy

In this section, 17 test cases are selected from *SPEC CPU2000* and input files in `train` data set are used. `Train` data set is designed for profile-driven compiler optimizations and can be viewed as the representative data set of `ref` data set. For each test case, identical input file is used for configuration search in the proposed software design flow and performance evaluation in experiments. Therefore, the obtained *expansion schemes* are optimized for the input file in each test case. In Figure 2.9, the improvements in terms of miss rate reduction ratio (as compared with the miss rate of *expandable cache*) are first reported. In average, our proposed method leads to 14.74% improvement. In following analysis, we focus on the test cases with larger improvement ratios (more than 10%).

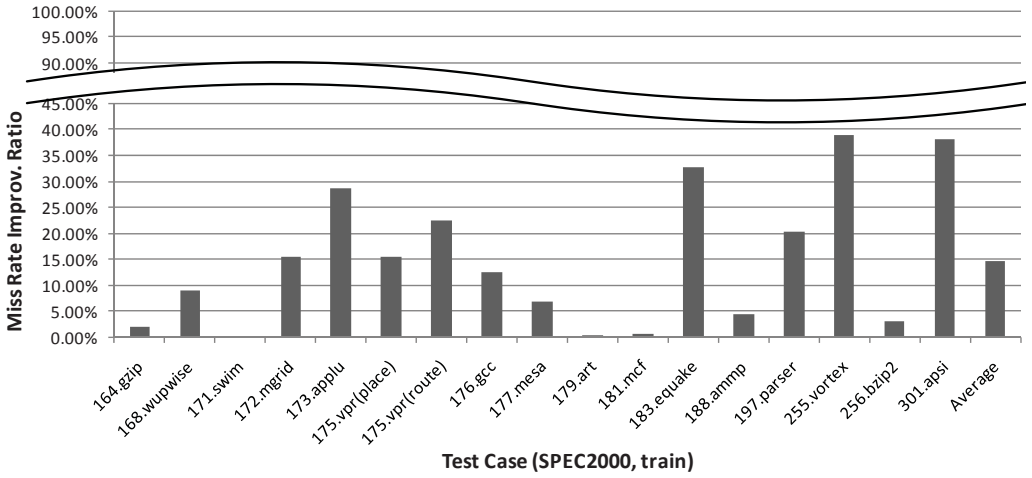



Figure 2.9: D-Cache Miss Rate Improvement Ratio (as compared with 2-Way Set-Associative Cache)

In test case, `parser`, a large number of sequential data segments are spread out all over the memory address space. Each of these segments is mapped to a number of consecutive sets in data cache. During the execution, multiple data segments are accessed simultaneously. When these data segments are mapped to cache sets, they may overlap with one another. A single *expansion scheme* proposed in the original *expandable cache* is insufficient in resolving the confliction among these data segments. On the contrary, our proposed cache structure is capable of expanding conflicting sets to temporarily unused sets through various *expansion schemes*. Similar execution condition can also be observed in test cases, `equake` and `vortex`.

We also observed that compound data structures are widely used in test case, `vortex`. When compound data structures are used, each data object may be mapped to multiple sets in data cache. An example is depicted in Figure 2.10. In Figure 2.10, a compound data type named INFO is defined

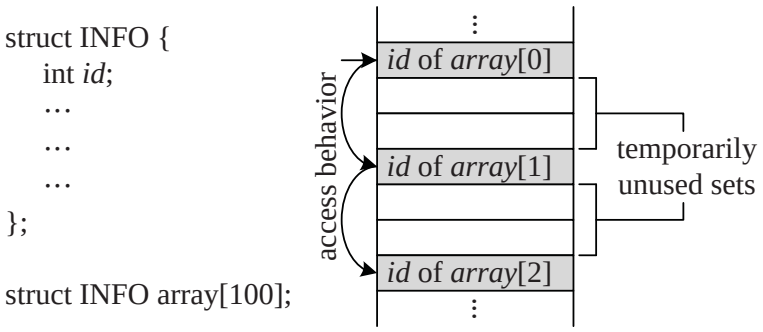



Figure 2.10: Access Behavior of Compound Data Structure in Data Cache

and an array is declared. Assume that the size of each data object is three times the size of a set. Then, the data field, *id*, of each data object is mapped to sets that are not consecutive. If the data field, *id*, of all data objects in the array needs to be accessed, the sets in gray color will be accessed and the sets in white color will remain unused. In this case, flexible *expansion scheme* is required to effectively utilize these unused sets. Compound data structures are also widely used in test cases, `vpr` and `gcc`. In these cases, our proposed method outperforms *expandable cache* because the access behaviors of compound data structures are considered. In test cases, `applu` and `apsi`, multi-dimensional arrays are intensively used. The accesses to these arrays can result in irregular access behaviors in data cache if the accesses are not based on the index of the last dimension. In these cases, the software design flow proposed in this work is capable of finding suitable *expansion schemes* to reduce cache misses.

Using the energy consumption of 2-way set-associative cache as baseline, the energy improvement ratios of direct-mapped cache, original *expandable cache*, and our proposed cache structure are presented in Figure 2.11. The

energy improvement ratio is computed as $\frac{\mathbf{E}.\ of\ 2-Way\ Set-Assciative\ Cache}{\mathbf{E}.\ of\ Tested\ Cache\ Structure}$. In average, all three tested cache structures have lower energy consumption as compared with 2-way set-associative cache. For direct-mapped cache, unless its miss rate is much larger than that of 2-way set-associative cache, it results in lower energy consumption. In average, our proposed cache structure achieved 22.86% energy improvement as compared with 2-way set associative cache. As compared with original *expandable cache*, our energy improvement ratio is 5.62% higher.

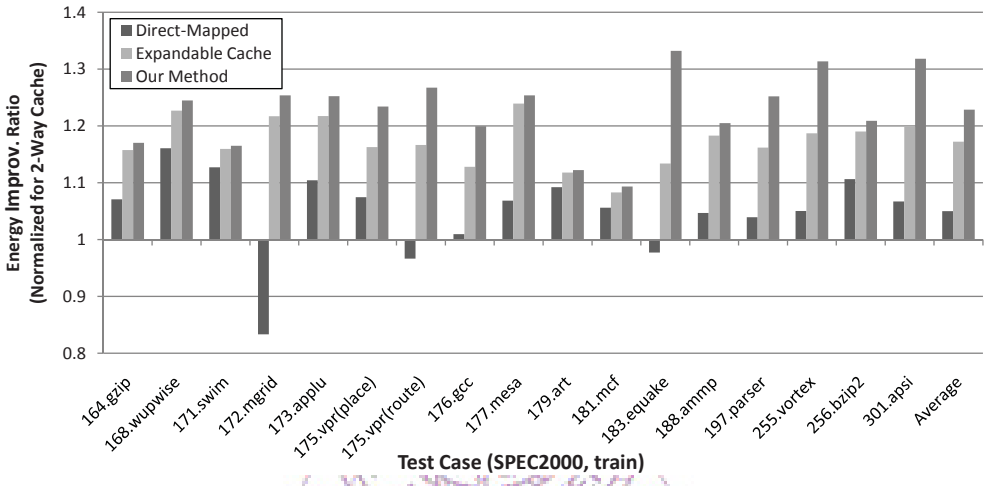



Figure 2.11: D-Cache Energy Improvement Ratio (as compared with original *Expandable Cache* [2])

Next, the overhead of our proposed method is discussed. First, the insertion of configuration instructions causes the increase in static instruction count. Moreover, additional execution cycles are required due to *expansion scheme* re-configuration. The overhead is summarized in Table 2.2. It shows that the overhead is insignificant.

Table 2.2: Overheads of Our Cache Structure

| **Overhead (in Average)** | **Value** |
|---|---|
| Static Instruction Count | 1.63% |
| Cycle Count | 0.0618% |

### 2.4.3 Experiment Results for Test Cases with Different Input Data

In this section, experiments are conducted to observe the miss rate of the proposed cache structure when the input file of each test case is not known. Test cases with multiple input files in `ref` data set are selected for experiments. For each test case and its input files in `ref` data set, two sets of *expansion schemes* are tested. The *expansion schemes* in the first set are determined by the input files in `train` data set. That is, no matter which input file in `ref` data set is tested, an identical set of *expansion schemes* is used. On the contrary, the *expansion schemes* in the second set are determined using dedicated input files in `ref` data set. That is, for each test case and each of its input file, a set of dedicated *expansion schemes* is used. Through experiments, we want to discuss whether dedicated profiling is required when the input of an application is changed. The experimental results for the selected test cases are presented in Figure 2.12. For each test case, all input files in `ref` data set are tested and the miss rate reduction ratios are reported. Our results are compared with those of *expandable cache*.

In test case `gzip`, as shown in the leftmost column in Figure 2.12, the experimental results of our method are worse than the results of *expandable cache* when fixed *expansion schemes* are used. A further analysis on the

data access behavior of `gzip` explains the reason. During the execution of `gzip`, sequential data segments with various sizes are created and accessed. These data segments determines the layout of memory space and are highly dependent on the input files. Therefore, when fixed *expansion schemes* are used in our method for different input files, the proposed cache structure cannot correctly expand busy sets to unused sets. In this case, dedicated profiling is required for each input file.

In test case `gcc`, as shown in the second column of Figure 2.12, the experimental results of our method are at least 5% better than the results of *expandable cache* no matter whether fixed or dedicated *expansion schemes* are used. As discussed in Section 2.4.2, compound data structures are widely used in this test case. When data objects of a dedicated data structure are accessed frequently in a program routine, the access behavior of data cache do not change much even when different input files are tested. In other words, the access behavior of data cache in this case is primarily determined by data structure and program routine. Different input files only changes the execution ratio of each program routine. Similar experimental results can also be observed in test case `vortex` in the third column of Figure 2.12. For these test cases, the *expansion schemes* determined by the representative input files in `train` data set can be used for different input files.

In test case `bzip2`, as shown in the last column of Figure 2.12, the experimental results of our method are slightly better than those of *expandable cache* when fixed *expansion schemes* are used. In test case `bzip2`, lots of small data segments with similar sizes are created and accessed. The access order and access frequency of these data segments are dependent on input

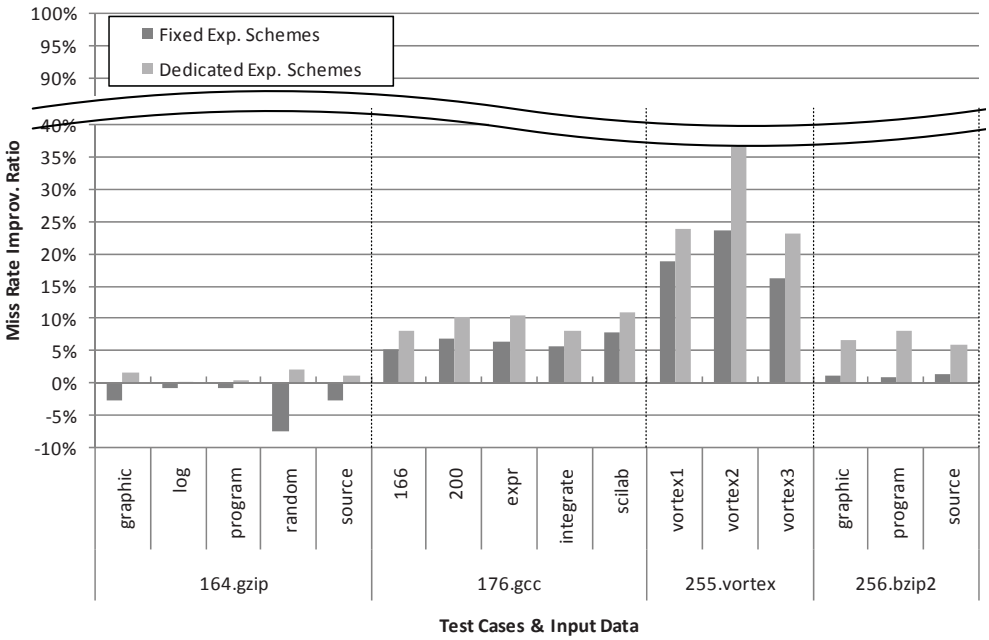



Figure 2.12: Miss Rate Improvement Ratios for Test Cases with Multiple Input Data (SPEC2000, ref)

files. Unlike test case `gzip`, the data segments in `bzip2` are located more regularly in memory space. Although the run-time access behaviors of these data segments are quite diverse for different input files, the memory layout of these data segments are similar. This property is beneficial for our proposed cache structure. Therefore, our method can still provide comparable miss rate reduction as compared with *expandable cache* when fixed *expansion schemes* are used.

From our experiments, some observations can be drawn. First, when compound data structures are widely used in an application, the data cache access behavior of that application is primarily determined by data structure and program routine. In this situation, fixed *expansion schemes* are enough

for different input files. Second, the memory layout of data is also important to our proposed cache structure. As long as the memory layout of data does not change dramatically when different input file is tested, fixed *expansion schemes* are able to provide comparable improvement on miss rate reduction as compared with *expandable cache*. Finally, if the memory layout of data is highly dependent on input file, dedicated profiling is required for different input file.

# Chapter 3

# Thermal-Aware Memory Mapping in 3D Designs

System in package (SIP) provides a cost-effective solution for large-scale integration [23]. 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 [24]. 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 [25] [26] [27] [28] [29]. Though SIP technology provides extremely high capacity for circuit integration, it suffers severe thermal stress because of three dimensional stacking of ICs [30]. Thermal stress will induce variation of DRAM retention time and reliability problem [31].

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 [32], throttling techniques [33], and non-DVS localized ther-

mal management [34] 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. [35] 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 [36] is proposed from compiler-level point of view which distributes computations to different functional units so that the hotspots are prevented.

Except [37], none of previous research addressed thermal and energy problem for 3D memory design. In [37], 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.

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 logic block but also the physical location during memory mapping, second, the changing temperature of a physical block during execution of programs. In this chapter, 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 this chapter is organized as follows. In Section 3.1, motivation of this work is presented. Section 3.2 describes our system model and problem definition. In Section 3.3, 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 3.4.

## 3.1 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 3.1. 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,

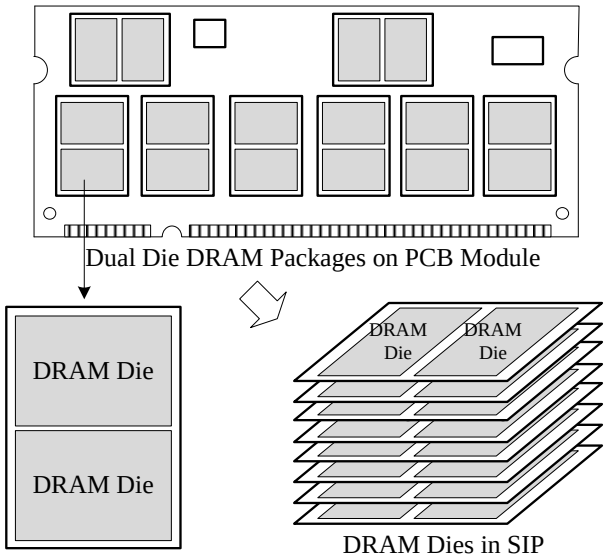



Figure 3.1: DRAM Packages on PCB and DRAM Dies in SIP Design

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.

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 3.2 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 3.2(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 3.2(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 3.2(c)-(e). Figure 3.2(c) shows a straightforward mapping (*Mapping A*) where two banks at the same relative position denoted as $A$, $B$, $C$, $D$ are accessed simultaneously. Figure 3.2(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 3.2(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 [38] is presented in Figure 3.2(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

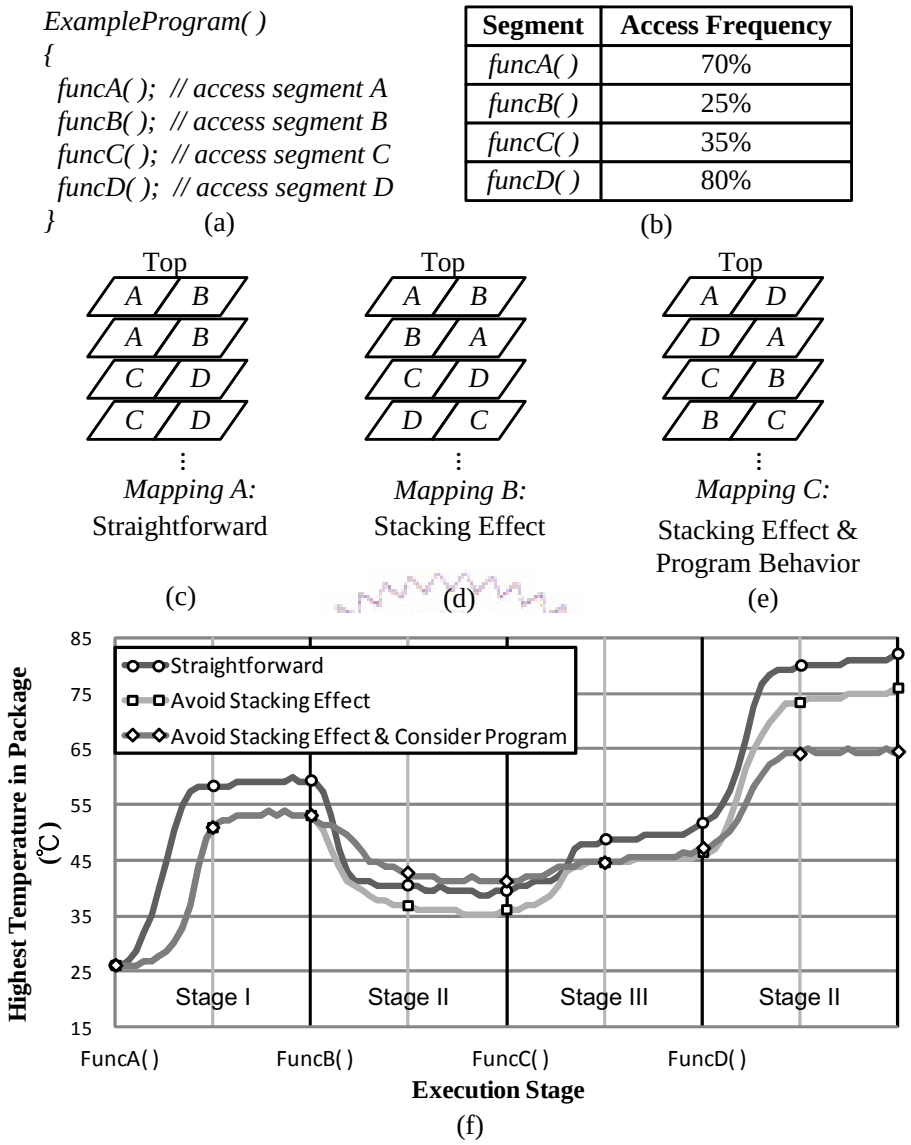



Figure 3.2: (a) Example Program; (b) Access Frequency; (c) *Mapping A*; (d) *Mapping B*; (e) *Mapping C*; (f) Simulation Result

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.

## 3.2 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 3.3 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

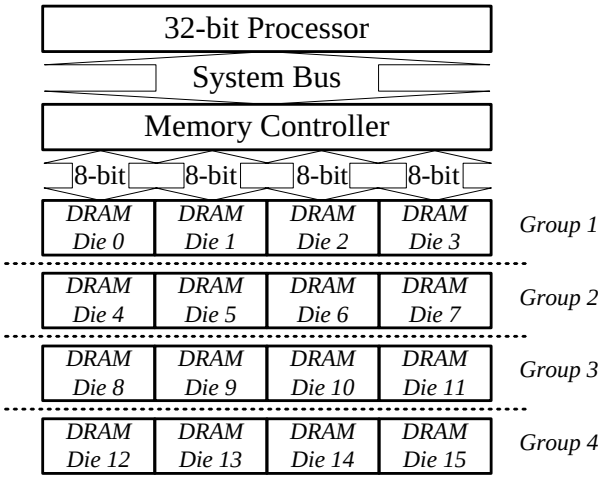



Figure 3.3: Memory System

word capacity of one *group*.

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 3.4(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 3.4(b). For each memory access, *Control & Pre-charge Circuits* block is always triggered. This block contains control, error correction and pre-charge 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 3.5 as an example using the same system config-

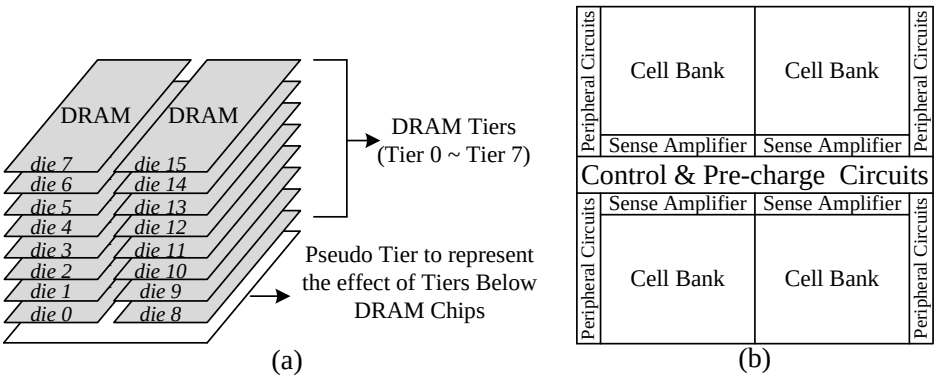



Figure 3.4: (a) SIP Model; (b) Floorplan of a Typical DRAM chip

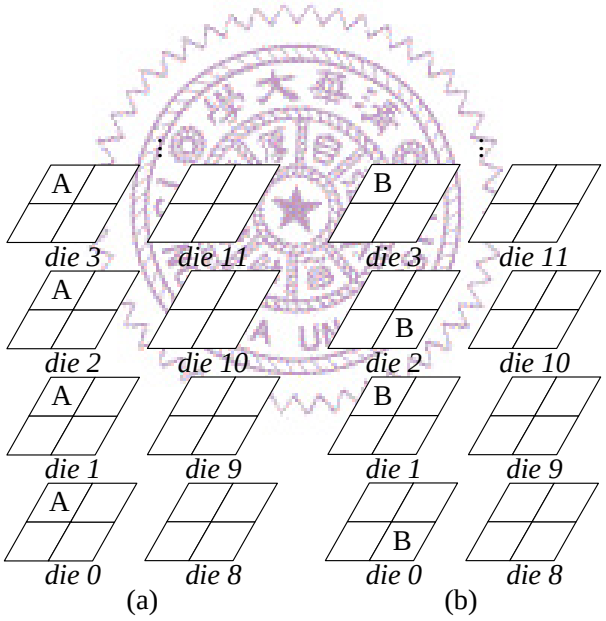



Figure 3.5: Example for Memory Access

uration as shown in Figure 3.3 where 4 dies form a *group*. In Figure 3.5(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 3.5(a) shows that banks denoted as $A$ in the same relative position form a *set*. In Figure 3.5(a), 4 dies form a *group* and there are 4 *sets* in a *group*.

However, in a stacked SIP design, Figure 3.5(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 3.5(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 chapter, 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 3.6 is proposed. The first step, *Determination of Candidate Configurations* is, for given parame-

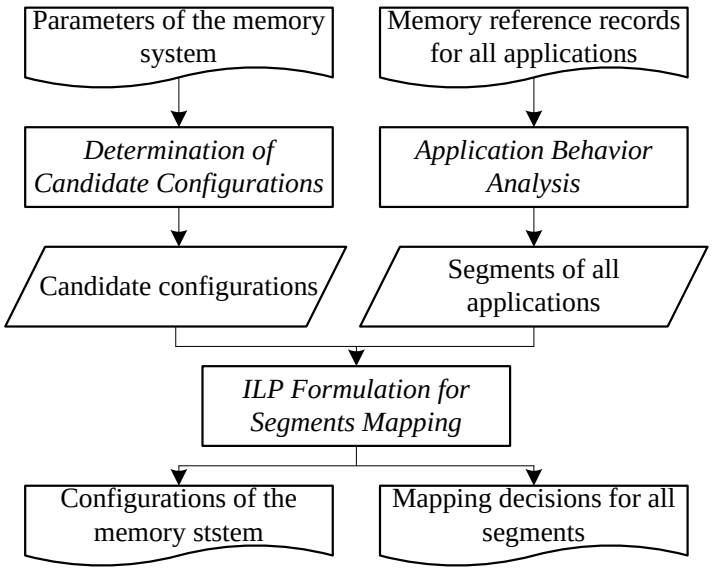



Figure 3.6: Overall Flow

ters of a memory system, to find candidate memory configurations (in Section 3.3.1). 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 3.3.2). According to the candidate configurations and *segments* obtained, the last step, *ILP Formation for Segments Mapping*, is an ILP formulation to perform mapping so that the maximum temperature is minimized (in Section 3.3.3).

## 3.3 Thermal Driven Memory Address Mapping Algorithm

Before we present our mapping algorithm, we first review some terms defined in Section 3.2.

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

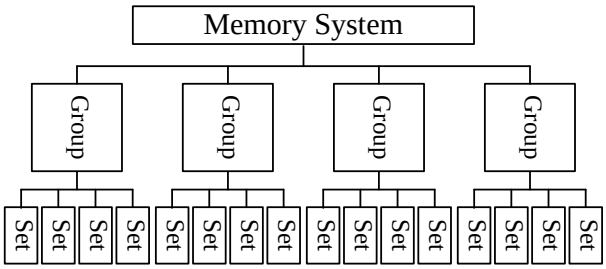



Figure 3.7: Hierarchical View

*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 bandwidth of a DRAM die, $\#bandwidth\_die$, the size of a DRAM die, $\#bit\_die$, and the bandwidth of system bus, $\#bandwidth\_system$. $\#bandwidth\_system/\#bandwidth\_die$ 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 $\#bit\_die/(\#bandwidth\_die \times \#bank)$.

### 3.3.1 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 3.3 and 3.4 as an example. In this example, because of

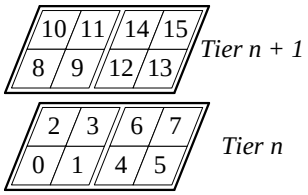



Figure 3.8: Memory Banks of Tier n & Tier n + 1

$\#bandwidth\_system = 32$ and $\#bandwidth\_die = 8$, 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 3.7 gives the hierarchical view of the system.

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 3.4(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 $\#die\_on\_tier \geq 2$.

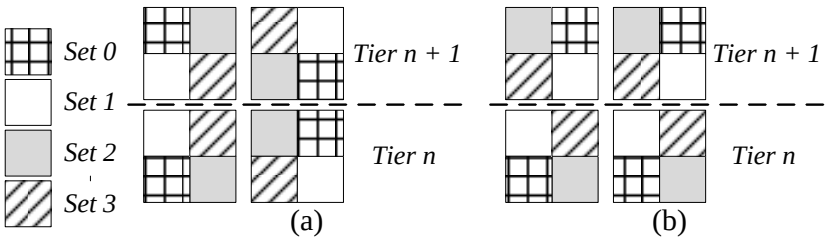



Figure 3.9: (a) *Configuration I*; (b) *Configuration II*

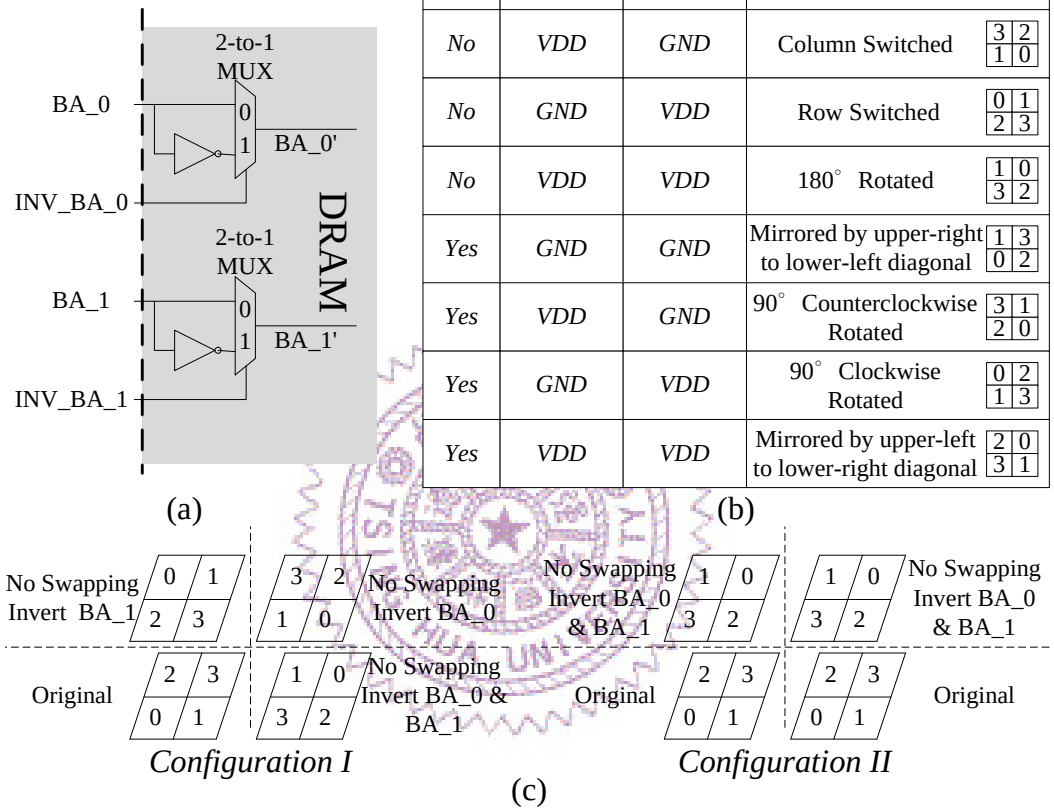
For example, suppose there are two dies on a tier as shown in Figure 3.8. Banks 1, 3, 4 and 6 are in the middle area of the tier and therefore have worse thermal behavior than banks 0, 2, 5 and 7. Second, accessing banks of different dies at the same vertical position will result in undesirable thermal 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 3.8 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 3.9 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 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 3.10(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




| 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 |

Figure 3.10: (a) Re-mapping Logic; (b) Re-mapping Table; (c) Example

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 3.10(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 3.10(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 (3.1) considers different ratios of read and write access where $\alpha$ represents the ratio of read access to total access and $(1-\alpha)$ the ratio of write access to total access.

$$Power_{Operating} = Power_{Read} \times \alpha + Power_{Write} \times (1-\alpha) \qquad (3.1)$$

Next, with different access frequency to a *set*, $f$, Equation (3.2) is defined for the average power.

$$Power_{Avg} = Power_{Operating} \times f + Power_{Standby} \times (1-f) \qquad (3.2)$$

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 chapter, HotSpot 4.0 [38] 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 $f$. This term will be used to define the cost function in Section 3.3.3.

### 3.3.2 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 3.11. 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.

In *Application Behavior Analysis* stage, a first-step merging is performed to merge neighboring *segments*. Neighboring *segments* which have similar behaviors are merged. 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*.

### 3.3.3 ILP Formulation for Segment Mapping and Group Configuration

After we find candidate *configurations* for each *group* and analyzed the behaviors of programs, how to select the most suitable *configuration* for each *group* and how to map each memory *segment* to an appropriate *set* remain to be solved. An ILP formulation is presented to solve these problems simultaneously. First, the terms used in our formulation are introduced as follows:

$Config_{x,y}$ 1 if *configuration* $x$ is selected for *group* $y$.

$Segment_{i,Set_{x,y,z}}$ 1 if *segment* $i$ is mapped to $Set_{x,y,z}$ where tuple $(x, y, z)$ refers to the $z$ *set* in *group* $y$ with *configuration* $x$.

$Cost_{i,Set_{x,y,z}}$ Cost of mapping *segment* $i$ to $Set_{x,y,z}$. The cost model will be presented in later paragraph.

$UNIT\_SET\_SIZE$ a constant to denote the size of a *set*.

$SetSize_{x,y,z}$ the size of $Set_{x,y,z}$.

$SegmentSize_i$ the size of *segment* $i$.

Then, the ILP formulation for our problem can be given as

$$Minimize \sum_i \sum_{x,y,z} Cost_{i,Set_{x,y,z}} \times Segment_{i,Set_{x,y,z}} \tag{3.3}$$

```
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();
```

Figure 3.11: Algorithm for *Behavior Analysis Algorithm*

subject to

$$\sum_{x} Config_{x,y} = 1, \ \forall \ y \tag{3.4}$$

$$\sum_{x,y,z} Segment_{i,Set_{x,y,z}} = 1, \ \forall \ i \tag{3.5}$$

$$UNIT\_SET\_SIZE \times Config_{x,y} = SetSize_{Set_{x,y,z}}, \quad \forall \ x, \ y, \ z \tag{3.6}$$

$$\textstyle\sum_i SegmentSize_i \times Segment_{i,Set_{x,y,z}} \leq SetSize_{Set_{x,y,z}}, \quad \forall \ x, \ y, \ z \tag{3.7}$$

The $Cost_{i,Set_{x,y,z}}$ (the detail of computing $Cost_{i,Set_{x,y,z}}$ will be explained later) represents the temperature cost when *Segment* $i$ is mapped to $Set_{x,y,z}$. The objective is to minimize the mapping cost. Equation (3.4) guarantees each *group* has exactly one *configuration*. Equation (3.5) is required to make sure each *segment* maps to only one *set*. Equation (3.6) ensures that if $Config_{x,y} = 1$ (*configuration* $x$ is selected for $y$ *group*), the size of all *sets* under $x$ *configuration* for $y$ *group* is equal to *UNIT_SET_SIZE*. Equation (3.7) requires that all *segments* map to $Set_{x,y,z}$ can not exceed the size of $Set_{x,y,z}$.

Let the number of candidate *configurations* be $X$, the number of *groups* be $Y$, the number of *sets* in each *group* be $Z$, and the number of *segments* to be mapped be $S$. The number of variables in the formulation is $O(X \times Y \times Z \times S)$. $X$, $Y$ and $Z$ are fixed values determined by system parameters. Therefore the number of variables in the formulation is determined by $S$. Since the number of *segments* produced in the stage of *Application Behavior Analysis* may be hundreds to thousands, the number of variables in the formulation can be too large to be solved by most ILP solvers effectively. To limit the

number of *segments* in the formulation, a second-step *segment* merging is needed.

The detailed merging process is explained as follows. First, the values of the highest and the lowest access frequencies of all *segments*, which are denoted as $F_{high}$ and $F_{low}$, are reported by *Application Behavior Analysis* stage. A number of *buckets* are defined between $F_{high}$ and $F_{low}$. Let the number of *buckets* be $L$, the frequency limit of each *bucket* is defined as Equation (3.8).

$$F_{low}+b\times(\frac{F_{high}+F_{low}}{L}) \leq F_{bucket} < F_{low}+(b+1)\times(\frac{F_{high}+F_{low}}{L}),\ 0 \leq b < L \tag{3.8}$$

*Segments* with access frequencies fall in the same *bucket* are considered to have the same *access frequency levels* and are merged. Note that the size of a *segment* after the merging stage still needs be limited to alleviate possible fragmentation problems. For example, we set $1/8 \times UNIT\_SET\_SIZE$ as the size limit of a *segment*.

The value of $L$ determines the granularity of the merging scheme. A small $L$ leads to a coarse-grained merging and reduces the number of *segments* effectively. However, the mapping is less flexible and may result in higher temperature. The tradeoff between the number of *segments* in the formulation and the quality of the mapping result is discussed in Section 3.4.3.

Finally, we will explain how to compute $Cost_{i,Set_{x,y,z}}$. Let $freq(i)$ stands for the access frequency of *segment* $i$, $Cost_{i,Set_{x,y,z}}$ is defined as Equation (3.9), where $T(Set_{x,y,z}, f)$ is the temperature of $Set_{x,y,z}$ with $f$ frequency obtained

in simulation defined in Section 3.3.1.

$$Cost_{i,Set_{x,y,z}} = \gamma^{\frac{T(Set_{x,y,z},freq(i))-\alpha}{\beta}} \tag{3.9}$$

In the equation, $\alpha$, $\beta$ and $\gamma$ are all constant values. $\alpha$ is the base temperature and is defined as the minimal value of $T(Set_{x,y,z}, f)$. Therefore, $T(Set_{x,y,z}, freq(i)) - \alpha$ is always equal to or greater than 0. $\beta$ is a positive value to prevent the cost from growing too fast. It also implies that the order of the cost is increased by 1 if $T(Set_{x,y,z}, freq(i))$ is increased by $\beta$. $\gamma$ is a value greater than 1 and grows linearly with the number of *segments*. Because we have a min-max problem, to avoid the situation where few terms have very high values while most terms have low values, our cost function is defined to grow in exponential. A larger $\gamma$ can prevent our solution from the situation described above. When $\gamma$ is equal to or greater than the number of *segments*, the above-mentioned situation can be completely avoided.

## 3.4 Experimental Results

In this section, experimental results for different execution conditions are presented. The system parameters are listed in Table 3.1 where the power values are estimated based on the $I_{DD}$ values of a Hynix DDR400 512 Mb SDRAM chip [39]. We assume the system supports multiprogramming with Round-Robin scheduling and all programs run on the system are pre-loaded to memory. In Section 3.4.1, the program set is composed of MediaBench [40], PowerStone [41] benchmark suites and JM H.264/AVC CODEC [42]. The programs are duplicated to multiple instances to simulate systems with different memory utilization ratio. For experiments in Section 3.4.2, programs

from SPEC CPU2000 benchmark [21] suite are also included. SimpleScalar 3.0 [20] is used to generate memory reference records. lp_solve 5.5 [43] is used as our ILP solver. HotSpot 4.0 [38] 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*. Note 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.

In Section 3.4.1, a single-core system with 512B cache size is used to examine the effectiveness of our proposed memory mapping flow. Next, in Section 3.4.2, a multi-core system with 4KB/8KB cache size is used for experiment. For both experiments in Section 3.4.1 and Section 3.4.2, the cache architectures are direct-mapped and the value of $L$ is set to 12. The trade-off between the value of $L$ and the quality of mapping result is discusses in Section 3.4.3.

### 3.4.1 Experiments for Single-Core System

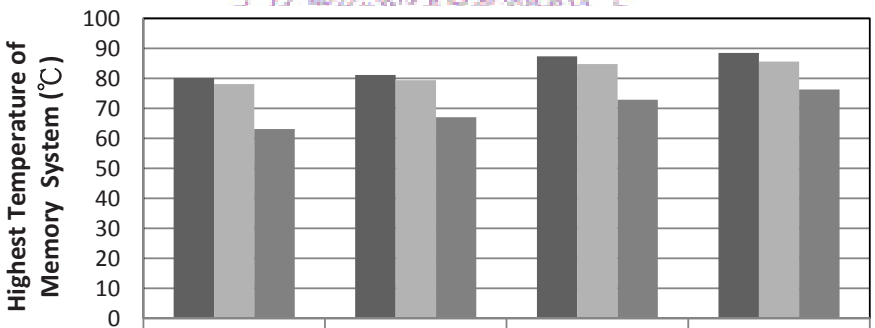
The experiments in this section demonstrate the effectiveness of our proposed memory mapping flow when main memory is accessed with high frequencies. A single-core system with 512B cache memory is used in this section. The size of the cache memory is intentionally reduced. In general, small cache size increases miss rate as well as access rate to main memory. This reduced cache size leads to higher cache miss rate and higher memory access rate.

Table 3.1: Parameters for Experiments

| System Parameter | Value |
|---|---|
| *#tier* | 8 |
| *#die_on_tier* | 2 |
| *#bank* | 4 |
| *#bandwidth_die* | 8 |
| *#bandwidth_system* | 32 |
| **DRAM Chip Parameter** | **Value** |
| Capacity | 512 Mb |
| Internal Clock Rate | 200 MHz |
| Voltage | 2.5 V |
| Max. Power | 0.75 W |
| Standby Power | 0.34 W |
| **Total Memory Size** | **1 GB** |

| **L1 Cache Architecture** | |
|---|---|
| Single-Core | Size: 512B<br>Organization: Direct-Mapped<br>Line Size: 16-byte<br>(32 Sets) |
| Multi-Core | Size: 4KB/8KB<br>Organization: Direct-Mapped<br>Line Size: 32-byte/64-byte<br>(128 Sets) |

| **Power Values** (mW) | | | |
|---|---|---|---|
| **Mode** | **Sense Amplifier** | **Cell Bank** | **Control & Pre-charge** |
| Standby | 36.5 | 18.2 | 121.2 |
| Active Read | 186.4 | 94.9 | 211.6 |
| Active Write | 217.6 | 114.6 | 245.7 |




| | 75%, 800MHz | 95%, 800MHz | 75%, 1.2Gz | 95%, 1.2GHz |
|---|---|---|---|---|
| M_1 | 80.2 | 81.2 | 87.4 | 88.5 |
| M_2 | 78.1 | 79.4 | 84.8 | 85.6 |
| M_ours | 63.1 | 67.1 | 72.9 | 76.3 |

Figure 3.12: Comparisons of the Highest Temperature for Single-Core System

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 leftmost column of Figure 3.12, when memory utilization ratio is 75%, M_ours has the temperature reduction by 17.1°C and 15.0°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 area of 4 bottom tiers. Then, when the memory utilization ratio is increased to 95% (the second leftmost column), less memory space is left. The improvement of our method is decreased to 14.1°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 second rightmost column), the temperature of all mappings is increased by 7.9°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 lead 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 14.5°C and 11.9°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 rightmost column), the temperature is reduced by 12.2°C and 9.3°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*.

### 3.4.2 Experiments for Multi-Core System

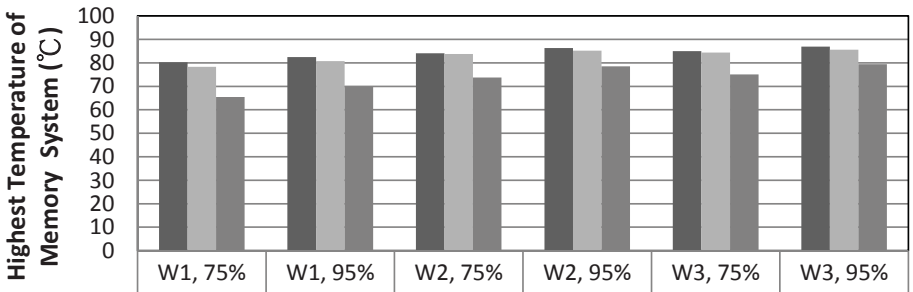
A multi-core system is assumed in this section. The objective of experiments in this section is to demonstrate the effectiveness of our proposed method when the memory system is accessed by multiple cores running different applications. Increasing the number of cores leads to higher access rate to memory system. To make the overall access rate to the memory system more reasonable, the cache memory size (L1 Cache) of each core is increased to 4KB in the first experiment of this section. In this chapter, the applications run on different cores are assumed to be independent to each other and hence data consistency is not considered. For each core, a set of programs are assigned as its workload. To avoid the memory access behavior

Table 3.2: Workload Combinations

| | **Core 1** | **Core 2** | **Core 3** | **Core 4** |
|---|---|---|---|---|
| *W1* | MediaBench | PowerStone | JM H.264 | `gzip` |
| *W2* | MediaBench + PowerStone | JM H.264 | `gzip` | `gcc` |
| *W3* | JM H.264 | `gzip` | `gcc` | `mcf` |

of each core to be similar, 3 more programs (`gcc, gzip, mcf`) from SPEC CPU2000 are added to our program set. In terms of program behavior, these three programs, as well as JM H.264/AVC CODEC, are much more complicated than other programs in our program set. Therefore, when a core is assigned with one of these programs as its workload, it will not be assigned with any other programs. Also note that these four complicated programs contain many consecutive memory reads and writes. Increasing the number of cores executing these complicated programs will largely increase the number of *segments* with high access frequencies. Since SimpleScalar does not support multi-core simulation, the memory access behavior of each core is first recorded individually. Then, the memory access behaviors of all cores are multiplexed to create the workload to the memory system. The system is set to contain 4 cores, and three types of workloads, $W1$, $W2$ and $W3$, are used in the experiments as listed in Table 3.2.

The experimental results are shown in Figure 3.13. In these experiments, the frequencies of the cores are set to 800 MHz. Similar to Section 3.4.1, for each workload combination, both 75% and 90% memory utilization ratios are tested. As shown in Figure 3.13, the maximum temperature of the system increases when the workload combination contains complicated programs. The average improvements of our proposed mapping flow for *W1*,

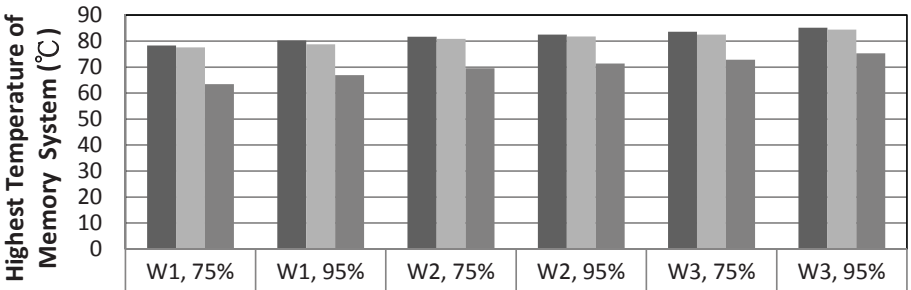



| | W1, 75% | W1, 95% | W2, 75% | W2, 95% | W3, 75% | W3, 95% |
|---|---|---|---|---|---|---|
| M_1 | 80.2 | 82.4 | 84.1 | 86.3 | 85 | 86.9 |
| M_2 | 78.3 | 80.7 | 83.8 | 85.2 | 84.4 | 85.6 |
| M_ours | 65.4 | 69.9 | 73.7 | 78.5 | 75.1 | 79.4 |

Figure 3.13: Comparisons of the Highest Temperature for Multi-Core System (L1 = 4KB)

*W2*, and *W3* are 12.75°C, 8.75°C, and 8.23°C respectively. An interesting phenomenon is that the improvement degradation between *W1* and *W2* are much larger than that between *W2* and *W3*. A further inspection on the mapping results explains the reason. For workload combination *W1*, the number of segments with high access frequencies is still small enough for our proposed method to map all these segments to tiers with better heat dissipating abilities. However, for workload combinations *W2* and *W3*, there are many segments with high access frequencies. Some of these segments are mapped to banks on middle tiers. Still, our proposed mapping flow provides considerable improvements as compared to M_1 and M_2.

One way to reduce the access frequencies to memory is to enlarge the cache of each core. When L1 cache size of each core is set to 8KB, the experimental results are shown in Figure 3.14. Similar to Figure 3.13, the maximum temperature of the system increases and the average improvements degrades as the workload combination contains more complicated programs. However,




| | W1, 75% | W1, 95% | W2, 75% | W2, 95% | W3, 75% | W3, 95% |
|---|---|---|---|---|---|---|
| M_1 | 78.3 | 80.2 | 81.7 | 82.5 | 83.6 | 85.2 |
| M_2 | 77.6 | 78.8 | 80.9 | 81.8 | 82.5 | 84.4 |
| M_ours | 63.4 | 66.9 | 69.6 | 71.4 | 72.8 | 75.3 |

Figure 3.14: Comparisons of the Highest Temperature for Multi-Core System (L1 = 8KB)

the improvement degradation becomes more stable. The average improvements of our proposed mapping flow for *W1*, *W2*, and *W3* are 13.58°C, 11.23°C, and 9.88°C respectively. The experiments indicate that the design of cache architecture is very important to 3D memory designs since it directly affects the memory access behavior.

### 3.4.3 Experiments for Segment Merging

In this section, the tradeoff between the value of $L$ and the quality of mapping result is discussed. The single-core system model introduced in Section 3.4.1 is used in this section. The frequency of the processor and memory utilization ratio are set to 800 MHz and 75% respectively. Different values of $L$ are tested. Experiments are performed on a Linux workstation with Intel Pentium 4 3.4 GHz CPU and 2 GB memory. For different values of $L$, the highest temperature and the computation time are summarized in Figure 3.15 and Table 3.3.

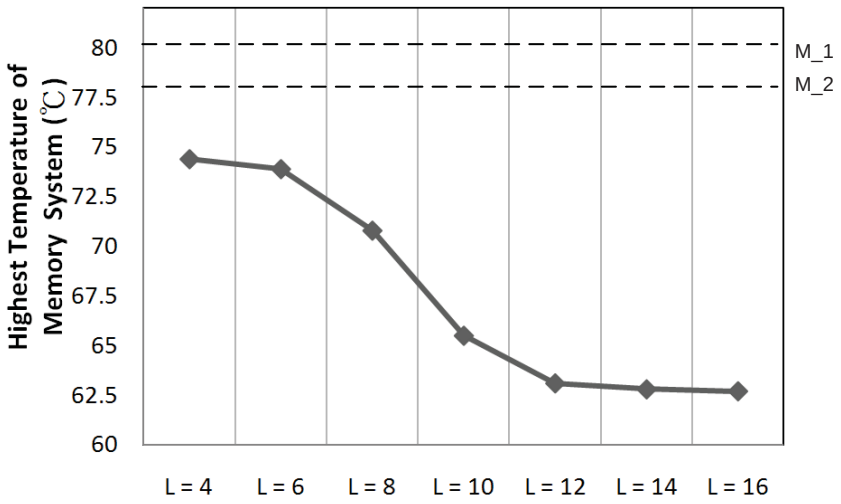



Figure 3.15: The Highest Temperature for Different Values of $L$

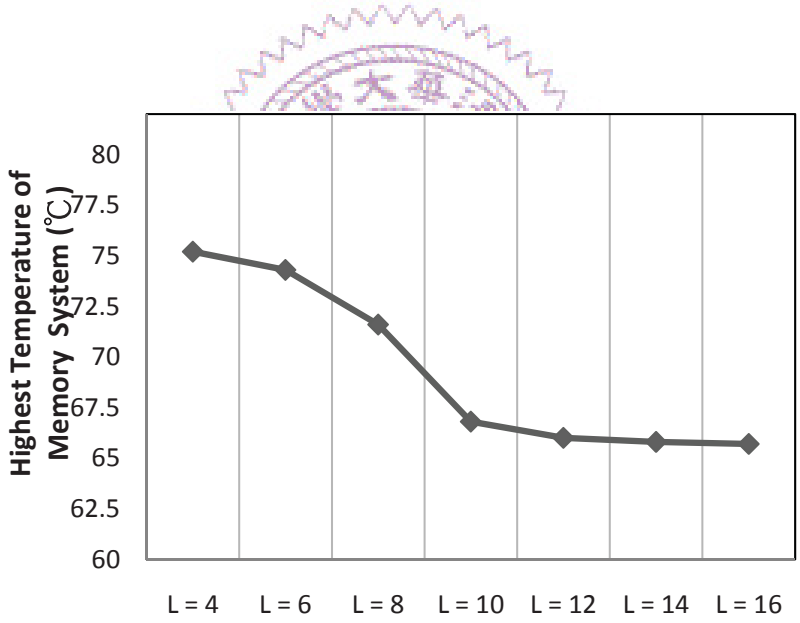



Figure 3.16: The Highest Temperature for Different Values of $L$ when the *Configurations* of all *Groups* are Restricted

Two dash lines which represent the highest temperatures of M_1 and M_2 (straightforward mappings) are depicted in Figure 3.15 for comparison. When the values of $L$ are set to 4 and 6, little improvement is observed. This is because the granularity of the merging scheme is too coarse-grained. *Segments* that have high access frequencies are merged with *segments* with relatively low access frequencies. Our proposed mapping flow can not accurately map all *segments* that have high access frequencies to physical locations with better heat dissipation. Therefore, the improvement is limited. As the value of $L$ increases, only *segments* with less difference in access frequency are allowed to be merged. Better mapping results can be obtained. When the value of $L$ is set to 12, *segments* are merged with sufficiently fine granularity. According to Figure 3.15, increasing the value of $L$ to be greater than 12 results in only small improvement. For the memory system model in our experiments, setting the value of $L$ to 12 is sufficient to provide good solutions.

A further analysis on the mapping results explains the relation between the value of $L$ and the organization of the memory system. For example, if the *configuration* of each *group* is restricted to *Configuration II* in our model, smaller value of $L$ may be sufficient. When a *group* is configured as *Configuration II*, all *sets* have identical thermal behaviors in that *group*. Therefore, the improvement due to the mapping from *segments* to *sets* in each *group* is minor. Figure 3.16 shows the experimental results for different values of $L$ when the *configurations* of all *groups* are set as *Configuration II*. Compared to Figure 3.15, Figure 3.16 shows the similar temperature reduction when $L$ is increased from 4 to 10. This part of improvement explains that the map-

Table 3.3: Computation Time

| $L$ | #segment | #var. | Computation Time |
|---|---|---|---|
| 4 | 93 | 2980 | 9 min 27 sec |
| 6 | 102 | 3268 | 12 min 3 sec |
| 8 | 117 | 3748 | 18 min 46 sec |
| 10 | 133 | 4260 | 25 min 19 sec |
| 12 | 157 | 5028 | 33 min 54 sec |
| 14 | 179 | 5732 | 45 min 12 sec |
| 16 | 204 | 6532 | 1 hr 3 min 7 sec |

ping from *segments* to *groups* is performed correctly. Comparing Figure 3.15 and 3.16 shows that, when the value of $L$ is increased from 10 to 14, further improvements can be obtained only in Figure 3.15. This is because the mapping from *segments* to *sets* in each *group* is allowed. These two experiments also show that the mapping from *segments* to *groups* provides major improvement in our proposed mapping flow.

For different values of $L$, the number of the *segments* after merging, the number of integer variables in the ILP formulation, and the computation time for the ILP solver are listed in Table 3.3. The number of integer variables in the ILP formulation increases linearly with the number of *segments*. Therefore, reducing the number of segments can effectively reduce the number of integer variables as well as the computation time. In general, ILP formulation can be solved in less than 1 hour when the value $L$ is less than 16.

# Chapter 4

# TSV Redundancy: Architecture and Design Issues in 3D IC

3D integration techniques are proposed as solutions to overcome the scaling limit [44] [45]. 3D technology provides many benefits including high density, high band-with, low-power, and small form-factor [46]. However, the increased power density and the complexity of fabrication process significantly degrades the reliability of systems. To address these problems, considerable amount of research has been conducted. It can be roughly classified into two categories. The first one focuses on possible execution-time failures of 3D ICs. Thermal problem and power/ground variation fall in this category [49]∼ [62]. The second category focuses on recovery mechanisms to cope with the faults caused in manufacturing processes [63]∼ [65].

In this chapter, the recovery of failed Through-Silicon Vias (TSVs) is discussed. TSV provides communication links for dies in vertical direction and is a critical design issue in 3D integration. Just like other components,

the fabrication and bonding of TSVs can fail [47] [48]. A failed TSV can severely increase the cost and decrease the yield as the number of dies to be stacked increases. To improve the yield, some recovery mechanism for faulty TSV is needed. A simple but effective solution is to add redundant TSVs to replace failed TSVs.

A redundant TSV architecture with reasonable cost is proposed in this chapter. Our proposed redundant TSV scheme is scalable and can be adjusted to fit the failure rate of TSV for different TSV processes and bonding technologies. Given the failure rate of TSV and the number of TSVs required, probabilistic models are presented to compute the number of redundant TSVs that should be allocated so that an expected assembly yield can be achieved. Related design issues and design flow are discussed. The proposed redundant TSV design can successfully recover most of the failed chips and increase the yield to 99% based on probabilistic models.

The rest of this chapter is organized as follows. First, previous work related to reliability issues of 3D IC is reviewed in Section 4.1. Then, in Section 4.2, the yield of TSV-based 3D IC is discussed. In Section 4.3, the proposed architecture for TSV redundancy and circuits for TSV testing are introduced. Then, in Section 4.4, the recovery rate and the number of redundant TSVs required for the proposed architecture are analyzed. Probabilistic model is used for evaluation. Next, the comparison between our proposed redundant TSV scheme and those in previous work is presented in Section 4.5. The design issues for timing and design flow are explained in Section 4.6.

## 4.1 Previous Work

Considerable amount of research has been conducted to improve the reliability of 3D IC. Due to the increased power density of 3D designs, thermal-aware physical design flow and related algorithms are intensively studied recently. In floorplan and placement stages, algorithms are developed to reduce power density [49]∼ [52]. Thermal problem in 3D IC can also be alleviated through the allocation of thermal vias [55]∼ [59]. By assigning thermal vias to high-power-density regions, thermal conducting problem can be improved. Algorithms for synthesis and optimization of power network for 3D IC are proposed to reduce power/groud variation [60] [61]. TSVs for power delivery can also be used for heat dissipation and therefore can be treated as thermal vias [58] [62].

Another category of research on reliability of 3D IC is the recovery of failed TSVs. A simple but effective solution to recover faulty TSVs is to use redundant TSVs. This idea has been realized in 3D DRAM designs [63], where for every 4 signals, 6 TSVs are allocated as a group. For each group, a switching box is used to select 4 TSVs for signal transfer. The advantage of this structure is that the delays of all signals are almost identical. This is an attractive property for DRAM designs. However, the cost is too expensive for ASICs since 50% additional TSVs are required. Another fault tolerance scheme that utilizes redundant TSVs aims at 3D network-on-chip (3DNoC) links [64]. In this work, TSVs are used to implement vertical interconnects between network switches on different tiers. For each network switch, 38 signal TSVs (32 data signals and 6 flow control signals) are allocated with 4

Table 4.1: Comparison of the Redundant TSV Schemes Proposed in Previous Work

| **Redundant TSV Scheme** | **#R_TSV** | **Hardware for TSV Recovery** | **Hardware for Testing** | **Design Type** |
|---|---|---|---|---|
| Switching Box in 3D DRAM [63] | 50% | Switching box circuits and e-fuses | Circuits for open-short tests, latches, and scan-chain circuits | Memory designs |
| Fault Tolerance Scheme for 3DNoC [64] | 10.53% | MUXs for signal redirection and e-fuses | 4 groups of scan-chain circuits controlled by FSM in each network switch | Designs with dedicated 3DNoC structure |
| The *Twins* Structure [65] | 100% | Not required | Not required | No constraint |

redundant TSVs. With this structure, the yield can be increased from 68% to 98% when 100K TSV links and 9.87 *Defect Per Million Opportunities* are assumed. This structure is effective. However, it is not good for other types of designs. The *twins* structure proposed by HRI [65] realizes each vertical interconnect as a pair of TSVs. In this redundancy scheme, the connectivity of a vertical link is maintained unless both TSVs of that link are failed. The yield can be largely increased. However, the number of TSVs is doubled. A comparison of these redundant TSV schemes are summarized in Table 4.1.

## 4.2 Failure Rate Analysis for TSV-Based 3D Designs

For different 3D integration technologies, different manufacturing processes are required and different yield rates can be observed. Currently, we cannot find a complete comparison of TSV failure rates among different bonding and process technologies. However, depending on the manufacturing processes required by different 3D integration technologies, causes for failure in

each process can be compared. In the following discussion, different bonding and process technologies are compared based the manufacturing processes required.

1. **Face-to-Face Bonding:** As compared with other 3D bonding technologies, face-to-face bonding leads to minimum changes of modern IC manufacturing processes. Since vertical interconnects are realized by bond pads above device and metal layers, no TSVs are required. Therefore, the failure caused by TSVs manufacturing can be avoided. The bonding process technology for face-to-face bonding is quite mature since similar technology has been used in modern IC package industry. In terms of yield, face-to-face bonding technology outperforms other bonding technologies that require TSVs for vertical interconnects. However, this bonding technology allows at most two layers of circuits to be integrated.

2. **Face-to-Back Bonding:** When face-to-back bonding technology is used for 3D integration, TSVs are required for vertical interconnects. Depending on when the processes for TSV take place, three process scenarios, via-first, via-middle, and via-last are proposed. The pros and cons of these process scenarios in terms of fabrication yield are summarized as follows.

   (a) **Via-First:** In via-first approach, TSVs are fabricated before *front end of line*. This means, when the processes of device layers take place, TSV are already presented. The processes of device layer may cause contamination or worsening of TSVs [48]. This problem

makes the electric behavior of TSVs in via-first approach more unstable as compared with other two approaches.

(b) **Via-Middle:** When the processes of TSVs take place between *front end of line* and *back end of line*, the contamination and the worsening problems can be avoided. However, the mechanical stress and the thermal expansion problems caused by the processes of TSVs may damage the components on device layer. To solve this problem, low-temperature processes and/or metallization material other than copper (ex: tungsten) are required. However, this may cause other defect features for TSV forming due to the immaturity of process technology.

(c) **Via-Last:** For via-last approach, processes of TSVs are performed after *back end of line*. Instead of *deep reactive-ion etching* (DRIE) method which is commonly used in via-first and via-middle approaches, laser drilling is usually required for TSV forming in via-last approach [48]. In general, laser drilling results in larger TSVs due to the size of laser beam. This aggravates mechanical stress and thermal expansion problems. Moreover, when laser drilling is used in TSV forming process, the sidewall of TSV cannot be as smooth and straight as that created by DRIE [48]. The silicon debris splashed from the melted wafer also leads to additional cleaning problem since this debris cannot be removed with traditional cleaning process [48]. The above mentioned facts increase the difficulty of deposition and metallization. The uniformity caused by shot-to-shot laser energy stability leads to the variation in the

dimension of TSVs [74]. Finally, the thermal effect generated by a laser beam can affect or damage the components on device layer [48].

In addition to the failure mechanisms mentioned above, failure of TSV is a primary reason that causes 3D designs to fail.

In general, the size of a TSV is much larger than other on-chip devices. This leads to certain unique defect features for TSV forming [66]. For example, during metallization process, void may be formed [68]. After the fabrication of TSVs, wafer thinning is performed. Presently, most 3D IC processes require each tier to be less than 100$\mu$m [48]. The surface roughness is an important factor to the yield of later TSV bonding stage. When the dies of consecutive tiers are stacked, the TSVs of the die in upped tier need to be bonded to the bond pads of the die on lower tier, as shown in Figure 4.1. Due to the alignment problem, a bond pad is required for each TSV [67] [68]. In addition to misalignments, TSVs can also fail in the soldering process [69]. Other failure mechanisms such as dislocation, process variations or mechanical stress also decrease the fabrication yield of TSVs.

Recent research has pointed out that misalignment and failures on bonding are primary failure mechanisms for TSVs [69]. Both of the technologies are very similar to the packaging methods used in current IC industry [48]. Although the exact failure rate of TSVs is still not clear, the failure rates of alignment and bonding can be used to perform a failure rate analysis for TSV. Considering the TSV diameter and the size of bond pads, the failure rate of a single TSV may ranges between $10^{-4}$ and $10^{-5}$ based on current packaging technology. This assumption roughly meets the yields of TSVs

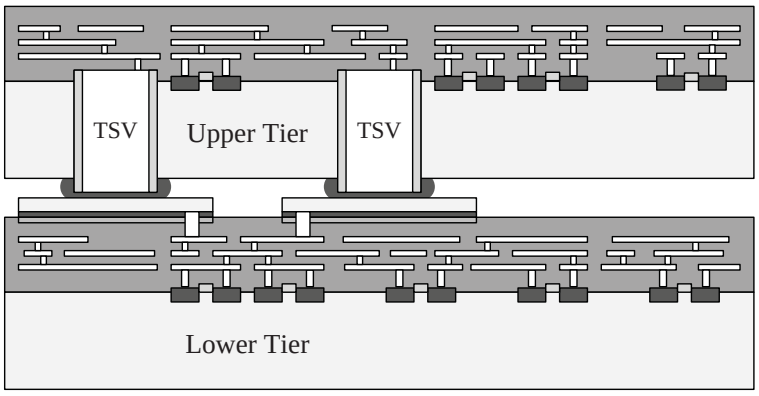
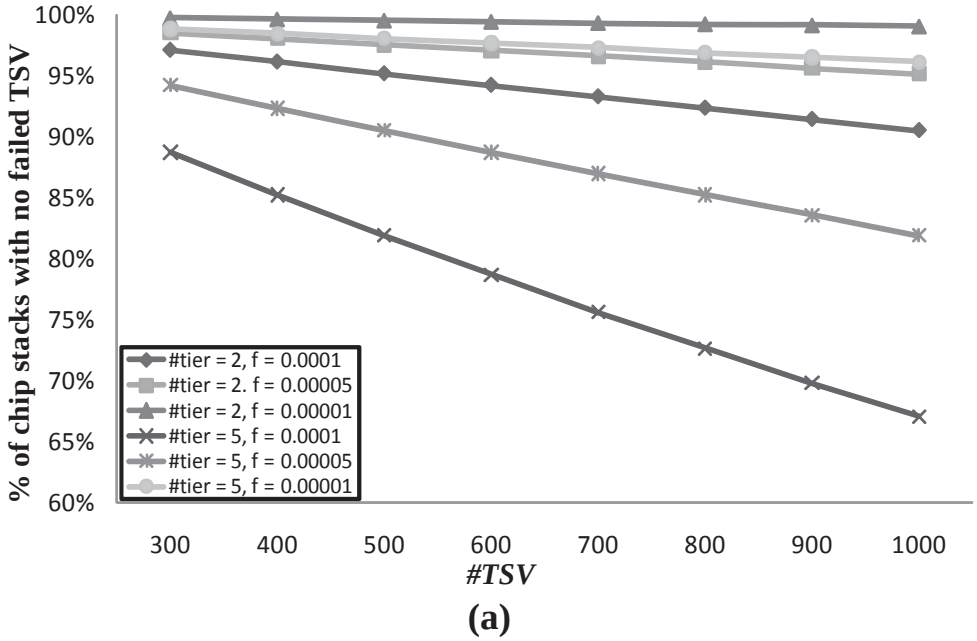



Figure 4.1: Bonding between TSVs and Bond Pads for 2-Tier 3D IC

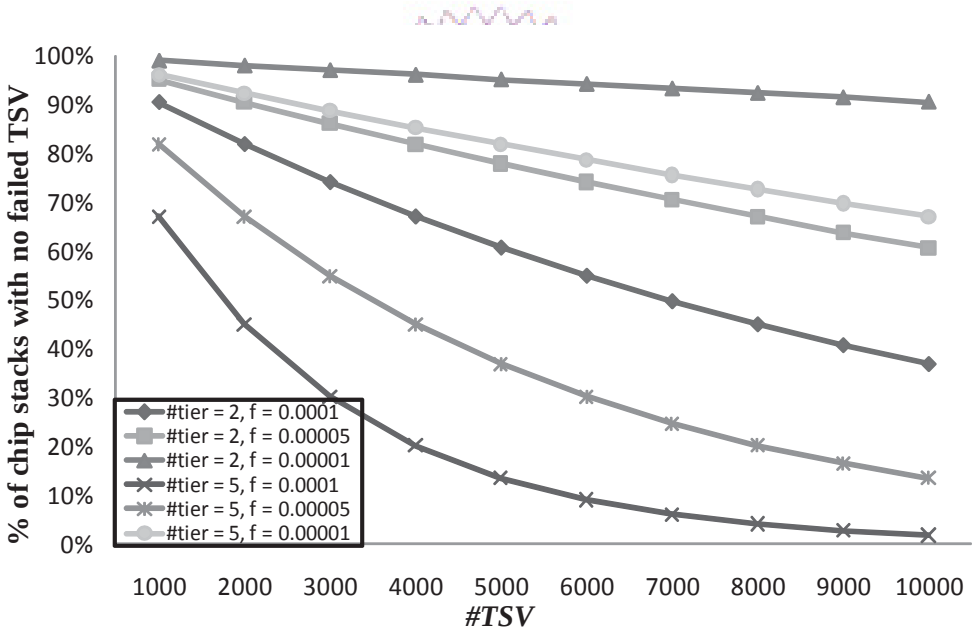
from the process technologies of HRI, IMEC and IBM [70] [71] [72].

According to the type of application, the number of TSVs in each tier can be quite different. For many-core processors or NoC-based designs, thousands of TSVs may be required in each tier. On the contrary, hundreds of TSVs may be sufficient for smaller IP-based designs. To consider both cases, we assume that the number of TSVs to be placed in a tier ranges from 1000 to 10000 for many-core processors or NoC-based designs, and 300 to 1000 for IP-based designs.

An analysis between failure rate and yield is given in Figure 4.2. Assume that all dies to be stacked are good dies. Thus, only the failure rate of TSV bonding needs to be considered. Let $f$ stands for the failure rate of bonding one TSV and $\#tier$ stands for the number of tiers to be stacked. The x-axis represents the number of TSVs to be placed in each tier ($\#TSV$). Since a good chip stack requires all TSVs to be successfully bonded, the binding yield can be computed as $(1-f)^{\#TSV\times(\#tier-1)}$.




**(a)**




**(b)**

Figure 4.2: Yield Analysis: (a) $\#tier = \{2, 5\}$, $\#TSV = 300{\sim}1000$; (b) $\#tier = \{2, 5\}$, $\#TSV = 1000{\sim}10000$

The analysis results for $f$ = {0.0001, 0.00005, 0.00001}, $\#tier$ = {2, 5}, and $\#TSV$ = {300∼1000, 1000∼10000} are shown in Figure 4.2. In Figure 4.2(a), $\#TSV$ ranges from 300 to 1000. When $\#tier$ equals 2 and 5, the average yields are 96.63% and 87.59%, respectively. When $\#TSV$ ranges from 1000 to 10000, the average yields degrade to 77.18% and 46.60% as shown in Figure 4.2(b). Even when $f = 0.00001$, the yield falls below 95% when $\#TSV \geq 6000$. From the analysis results in Figure 4.2, we can see that even if there are only two tiers, the yield may fall below 95% when $f$ or $\#TSV$ is large. When $\#tier$ is increased, the yields degrade further.

Note that the dies to be stacked are assumed to be good dies. The cost of discarding chip stacks due to failures on TSV bonding is very expensive. In fact, in most failed chip stacks, only a very small portion of TSVs are failed. If these failed TSVs can be recovered with reasonable cost, the yield can be largely improved. The redundant TSV design to be proposed in this chapter provides a solution to this problem.

## 4.3 Redundant TSV Architecture

### 4.3.1 Architecture Design

The proposed architecture for redundant TSV is depicted in Figure 4.3. For each TSV, 2 configuration MUXs are added to shift the signal to neighboring TSV when one TSV is failed. The TSVs are connected as a chain where the redundant TSV is placed at the last position of the chain. When no TSV is failed, all signals are transferred by original TSVs. When a TSV is failed, the signal of the failed TSV needs to be shifted. This in term causes all signals

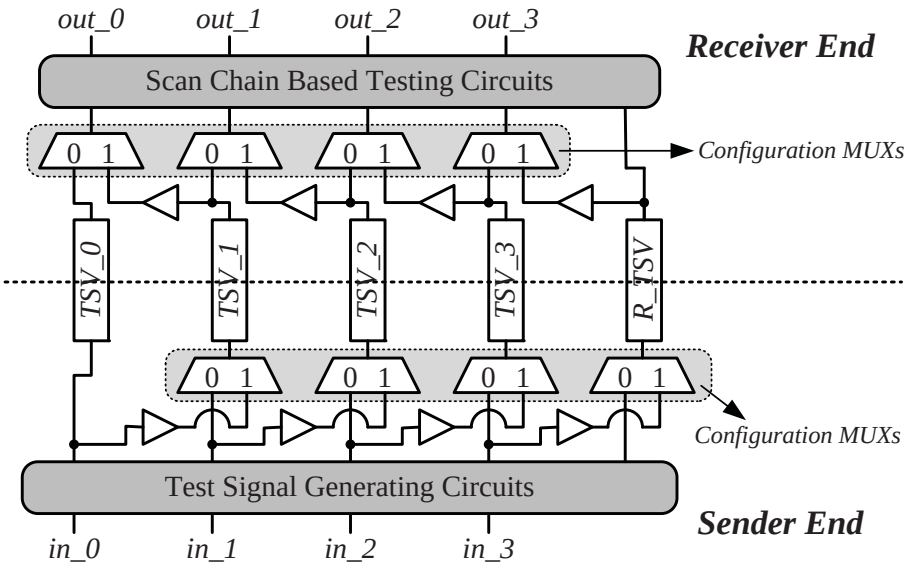



Figure 4.3: Architecture for Redundancy TSV

between the failed TSV and the redundant TSV to be shifted. For example, let *TSV_1* be failed. The selection inputs of configuration MUXs for *TSV_0*, *TSV_1*, *TSV_2* and *TSV_3* are set to 0, 1, 1 and 1, respectively. The signal paths after shifting are shown in Figure 4.4. When a signal is shifted, larger delay is introduced due to larger wire length and buffers. In this architecture, only one failed TSV can be recovered in each chain. Therefore, how to determine the number of TSVs in a chain so that an acceptable recovery rate can be achieved is an important design issue. For simplicity, the term *TSV-chain* is used to refer to the structure of the proposed redundant TSV architecture.

The configuration MUXs in the proposed architecture are connected to an e-fuse array. By default, all signals connect to the configuration MUXs are set to 0. The testing circuits to be explained in Section 4.3.2 is used to

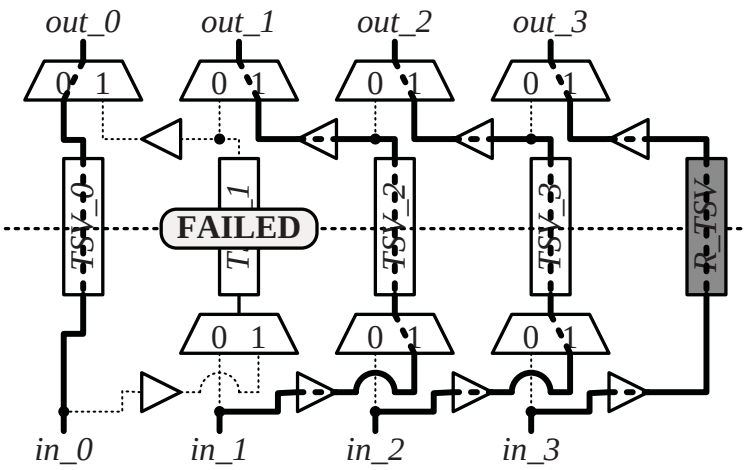



Figure 4.4: TSV Recovery Mechanism: TSV_1 is failed and TSV_1, TSV_2, and TSV_3 are shifted right one position

check the connectivity of each TSV. When the testing for TSV connectivity is done, the e-fuse array is programmed so that each configuration MUX receives an appropriate control signal.

The redundant TSV of each *TSV-chain* is not necessary to be located at the end position of the chain. By replacing the 2-1 configuration MUX with a 3-1 MUX, the redundant TSV is allowed to be located in the middle of the chain. The benefit of locating the redundant TSV in the middle of the chain is that the number of signals shifted can be reduced. However, the signal transferred by the redundant TSV will suffer larger delay due to a 3-1 MUX. Therefore, in this chapter, we restrict the redundant TSV to the end position of the chain.

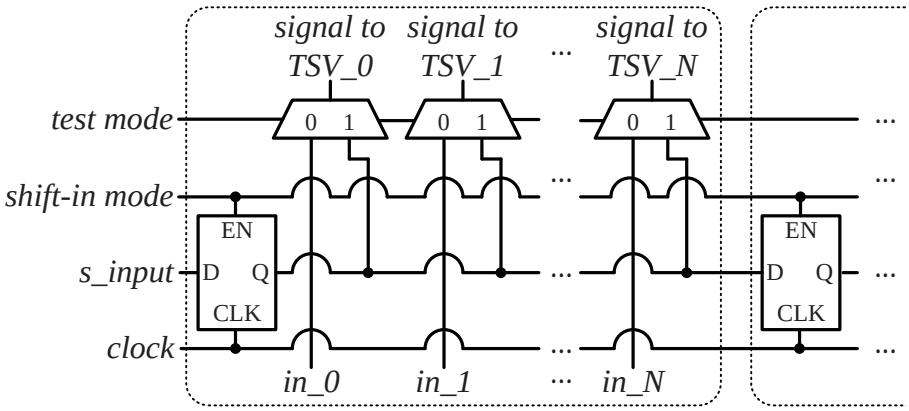



Figure 4.5: The Testing Circuits on the Sender End

### 4.3.2 Testing Circuits

The objective of the testing circuits is to test all TSVs and locate failed TSVs. The test results are used to program the selection inputs of configuration MUXs. The overall testing structure can be seen in Figure 4.3. On the sender end, circuits are added to generate test signals for each TSV, and on the receiver end, scan chain based circuits are added to capture the test signals. For each *TSV-chain*, both the signal TSVs and the redundant TSV are tested by our proposed testing structure. The testing circuits on the sender end and the receiver end are shown in Figure 4.5 and Figure 4.6, respectively. During the test, the selection inputs of the configuration MUXs on both sender and receiver ends are set to 0. The detailed testing mechanism is explained as follows.

For each TSV, both signals 0 and 1 are required to be tested. If both signals are transferred by a TSV correctly, the TSV is considered to be fault-less. Let the testing circuits on the sender end generate toggled signals in two

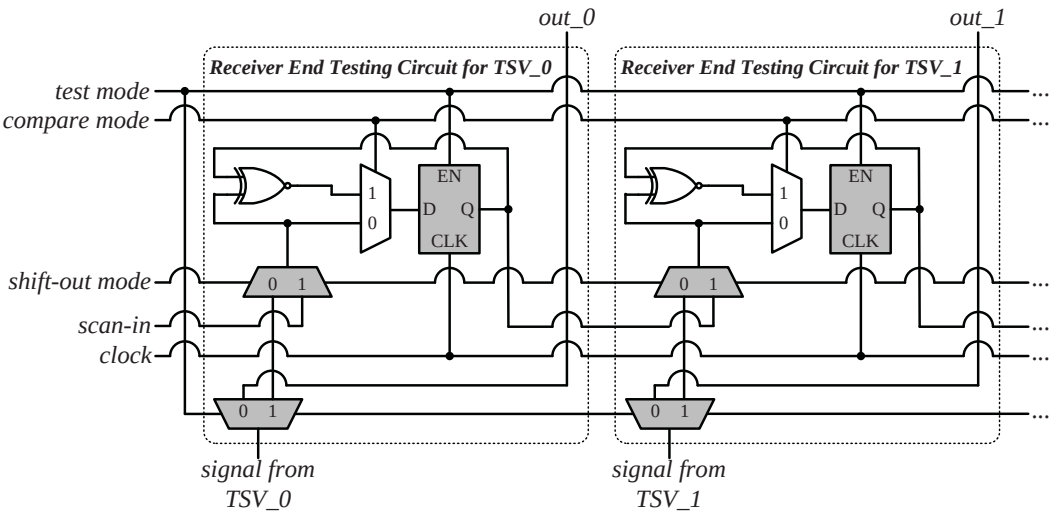



Figure 4.6: The Testing Circuits on the Receiver End

consecutive cycles. The testing circuits on the receiver end need to capture and record the signals sent by the testing circuits on the sender end, and shift-out the test results.

The testing circuits on the sender end shown in Figure 4.5 is described as follows. The MUXs on the upper part are used to select the signals to be transferred by TSVs in normal or test mode. When *test mode* = 0 in normal mode, signals are normal inputs and when *test mode* = 1 in test mode, signals are test inputs from flip-flops. The test signals in test mode are stored in flip-flops. Since no signals need to be captured on the sender end, it is not necessary to assign each TSV a dedicated flip-flip. Instead, we let multiple TSVs be grouped together to share the signal stored in one flip-flop. The number of TSVs in a group to be driven by one flip-flop is determined by the driving capability of the flip-flop. In the data shift-in mode, *s_input* is connected to a toggling signal which changes its value every

cycle. This allows the values stored in flip-flops of two neighboring groups to be different.

On the receiver end, the test results of all TSVs are used to program the selection inputs of the configuration MUXs in our proposed redundant TSV architecture. The circuits in Figure 4.6 are modified from traditional scan-chain circuits. The testing circuits for each TSV are surrounded by a dotted frame. In Figure 4.6, the testing circuits for two TSVs, *TSV_0* and *TSV_1* are depicted. For test circuit of a TSV, the devices in gray color are the components of traditional scan-chain circuits. The DEMUXs on the lower part are used to direct the signals transferred by TSVs. When *test mode* = 0 in normal mode, signals received by TSVs are normal signals. When *test mode* = 1 , signals received by TSVs are test signals. The MUXs in gray color are used to select the signals from TSV for testing or from neighboring flip-flop for shift-out.

The MUX and XNOR gates in front of the flip-flop provide the ability to perform an XNOR operation of the value received from TSV in the current cycle and the value stored in the flip-flop received in the previous cycle. If both signals are received correctly, the result of the XNOR operation in the second cycle should be 0. If the output of the XNOR gate is 1, one of the received signal must be incorrect. This indicates that the TSV under testing is failed. Note that the proposed circuits cannot identify a fault if both test signals are toggled. However, a failed TSV is similar to a broken wire. It is not likely to invert both signals 0 and 1. In general, for a failed TSV, the same values will be observed in two cycles.

Let the number of groups (flip-flops) on sender end be $n$ and the number

of flip-flops (i.e., the number of TSVs) on the receiver end be $m$. The overall testing process and the values of the control signals are shown as follows.

**Shift-In Stage I:**

$\{$*test mode* $= 0$, *compare mode* $= 0$, *shift-in mode* $= 1$, *shift-out mode* $= 0\}$ The signal *s_input* on the sender end toggles every cycle. The values of flip-flops on the receiver end are shifted-in in $n$ cycles.

**Capture Stage:**

$\{$*test mode* $= 1$, *compare mode* $= 0$, *shift-out mode* $= 0\}$ The test signals on the sender end are received by TSVs. The flip-flops on the receiver end capture the transferred signals in one cycle.

**Shift-In Stage II:**

$\{$*shift-in mode* $= 1\}$ In this stage, one more signal is shifted in on the sender end in one cycle. This causes the values stored in all flip-flops on the sender end to be toggled. This shift-in can be executed at the same cycle for **Capture Stage**.

**Compare Stage:**

$\{$*test mode* $= 1$, *compare mode* $= 1$, *shift-in mode* $= 0$, *shift-out mode* $= 0\}$ For each TSV, a toggled test signal is received. The signal is XNORed with the value stored in the flip-flop. The output of XNOR gate is then captured by the flip-flop in one cycle.

**Shift-Out Stage:**

{*test mode* = 1, *compare mode* = 0, *shift-in mode* = 0, *shift-out mode* = 1}

The values stored in the flip-flops on the receiver end are shifted out in $m$ cycles.

The shifted-out bit strings contain the test results of all TSVs. A 0 stands for a faultless TSV and a 1 for a failed one. For each *TSV-chain*, if more than one bit is set to 1, the whole chip stack needs to be discarded. Otherwise, the bit strings are used to program the e-fuse array so that each configuration MUX receives proper control value.

### 4.3.3 TSV Block and TSV-Chain

Due to manufacturing and physical design issues, TSVs are not recommended to be placed arbitrarily on a plane. From the aspect of manufacturing, a regular placement of TSVs improves the exposure quality of the lithographic process and therefore improves the yield. In real designs, TSVs are suggested to be placed regularly in *TSV blocks* which are determined in floorplan stage. Inside each TSV block, TSVs are arranged in a grid-based structure to satisfy the pitch constraint. Examples of TSV blocks are shown in Figure 4.7. Obviously, it is undesirable for a *TSV-chain* to contain TSVs of different TSV blocks due to long wires for signal shifting. Therefore, **a *TSV-chain* in our design is suggested to contain TSVs in the same TSV block**. Moreover, we let each TSV block contain only one redundant TSV. This means, for each TSV block, only one *TSV-chain* is defined. Nevertheless, in terms of recovery rate, the number of TSVs in a *TSV-chain* needs to be limited. In case the number of TSVs in a TSV block is too large for one *TSV-chain*, the TSV block needs to be partitioned to a number of smaller

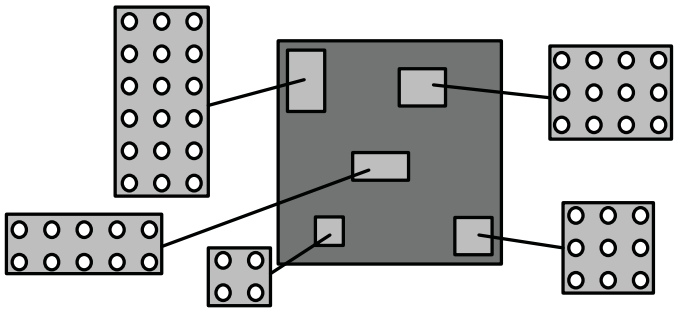


Figure 4.7: TSV Blocks

TSV blocks.

The design issues of our proposed *TSV-chain* are listed as follows:

- Determine the number of TSVs in each TSV block
- Determine the path to link the TSVs in a TSV block as a chain

The first problem is related to the recovery of a 3D design. In Section 4.4, an analysis based on probabilistic model is performed to answer this question. The second problem is related to the timing behavior of shifted signals. Discussions on timing issues and guidelines for *TSV-chain* design are presented in Section 4.6.

## 4.4 Recovery Rate Analysis

In this section, the relation between the number of TSVs in each TSV block and the recovery rate is analyzed based on probabilistic models. First, based on the failure rate of a single TSV, the expected number of TSVs that may fail in a tier can be computed. Next, for an expected number of failed TSVs

in each tier, the required number of *TSV-chains* as well as the size limit of each TSV block are discussed.

Let $F$ and $\#TSV_{tier}$ stand for the failure rate of a single TSV and the number of TSVs in a tier. The probability that exact $n$ TSVs are failed in a tier can be expressed as Equation (4.1) where $C_n^{\#TSV_{tier}}$ represents the number of combinations of $\#TSV_{tier}$ TSVs with $n$ of them failed and $F^n \cdot (1-F)^{\#TSV_{tier}-n}$ represents the probability of $n$ chosen TSVs are failed while other $\#TSV_{tier} - n$ TSVs are not.

$$P_{f\_tsv=n} = C_n^{\#TSV_{tier}} \times (F^n \cdot (1-F)^{\#TSV_{tier}-n}) \tag{4.1}$$

Next, the term $C\_Ratio_n$ is defined as the probability that the number of failed TSVs is no greater than $n$, including the faulty free condition (that is, $n = 0$). This can be computed by accumulating $P_{f\_tsv=i}$ for $0 \leq i \leq n$ and can be expressed as Equation (4.2).

$$C\_Ratio_n = \sum_{i=0}^{n} P_{f\_tsv=i} \tag{4.2}$$

As long as the value of $C\_Ratio_n$ is large enough, we can assume that the maximum number of failed TSVs in a tier is $n$. For example, when $n = 2$ and $\#TSV_{tier} = 500$, the value of $C\_Ratio_n$ is 99.998%. This means, when 500 TSVs are allocated in a tier, the probability that three or more TSVs are failed is less than 0.0020%. Assume that the maximum number of failed TSVs in a tier is 2. It covers 99.998% of all possible faulty free and faulty situations. In our recovery rate analysis, a high $C\_Ratio_n$ is expected. Given the desired value of $C\_Ratio_n$, the minimum value of $n$ increases with the value of $\#TSV_{tier}$. Assume that the value of $C\_Ratio_n$ is expected to be

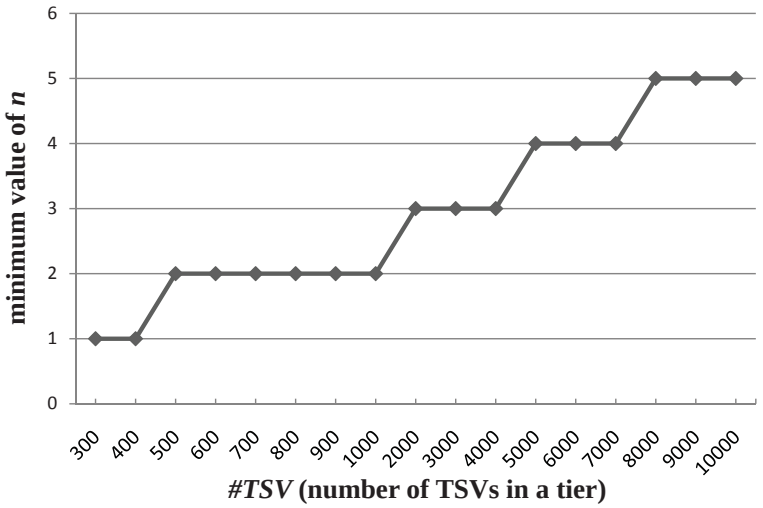



Figure 4.8: The Minimum Values of $n$ for $\#TSV_{tier} = \{300\sim10000\}$ ($C\_Ratio_n > 99.9\%$)

greater than 99.9%, the minimum value of $n$ for $\#TSV_{tier} = \{300\sim10000\}$ is depicted in Figure 4.8. In the rest of this section, for different values of $\#TSV_{tier}$, the maximum number of failed TSVs in a tier is assumed according to Figure 4.8 and is referred as $\#F\_TSV_{max}$.

As mentioned in Section 4.3, each *TSV-chain* is capable of recovering at most one failed TSV in a TSV block. As the number of TSVs in a TSV block increases, the probability that all failed TSVs can be recovered decreases. To achieve an expected recovery rate, the number of TSVs in each TSV block must be constrained. To simplify the analysis, we assume that the number of TSVs in all TSV blocks are identical. Let $\#B\_TSV$ stand for the number of TSVs in each TSV block. For a given value of $n$, we want to analyze the relation between $\#B\_TSV$ and recovery rate. For a given value of $\#TSV_{tier}$, the analysis for $n = \{1\sim \#F\_TSV_{max}\}$ is performed.

The number of combinations of $\#TSV_{tier}$ TSVs with $n$ failed TSVs can

be computed as $C_n^{\#TSV_{tier}}$. The number of combinations that all failed TSVs can be recovered by *TSV-chains* is referred as $\#Recoverable\_Comb$. The recovery rate discussed in this section is defined as

$$\frac{\#Recoverable\_Comb}{C_n^{\#TSV_{tier}}}.$$

When $n = 1$, only one failed TSV needs to be recovered. Since each TSV block contains one redundant TSV, the failed TSV can always be recovered. Thus, the recovery rate is 100%.

The recovery rate analysis for $n \geq 2$ is more complicated. Let the term $\#Block$ represent the number of TSV blocks in a tier. Under our assumptions, $\#Block$ can be computed as $\frac{\#TSV_{tier}}{\#B\_TSV}$. To successfully recover all failed TSVs, each failed TSVs must be located in different TSV blocks. That is, $n$ TSV blocks are selected from $\#Block$ TSV blocks. Each selected TSV block contains exactly one failed TSV. The number of combinations can be computed as

$$C_n^{\#Block}.$$

Inside each TSV block that contains one failed TSV, the failed TSV can be located at $\#B\_TSV$ possible positions. Therefore, the $\#Recoverable\_Comb$ can be computed as

$$C_n^{\#Block} \cdot (\#B\_TSV)^n.$$

For $\#TSV_{tier} = 500$, the value of $\#F\_TSV_{max}$ is 2. The relation between $\#B\_TSV$ and recovery rate for $\#TSV_{tier} = 500$ and $n = 2$ is shown in Figure 4.9. For different values of $\#B\_TSV$ that result in the same $\#Block$, only the smallest $\#B\_TSV$ is shown.

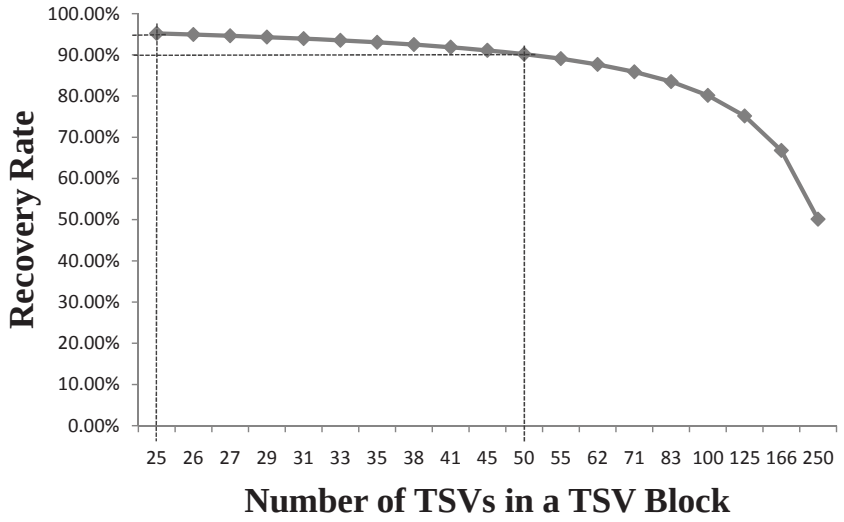



Figure 4.9: Recovery Rate when $\#TSV_{tier} = 500$, $n = 2$

According to Figure 4.9, to achieve 90% recovery rate, $\#B\_TSV$ needs to be no greater than 50. By limiting the number of TSVs in each TSV block to be less than or equal to 50, the recovery rate is greater than 90%. To achieve a higher recovery rate, the figure shows that with 95% recovery rate, the number of TSVs in each TSV block cannot be greater than 25. In real designs, if the number of TSVs in a TSV block is too large to satisfy the constraint, a large TSV block is partitioned into a number of smaller TSV blocks.

Similar analysis are performed for different values of $\#TSV_{tier}$. The limits on $\#B\_TSV$ for $\#TSV_{tier} = \{300{\sim}10000\}$ when $n = \#F\_TSV_{max}$ are shown in Figure 4.10. In Figure 4.10, a number of dash lines are depicted to denote the increase on $\#F\_TSV_{max}$. When $\#F\_TSV_{max}$ is given, the allowed maximum value of $\#B\_TSV$ increases linearly with the value of $\#TSV_{tier}$. When $\#F\_TSV_{max}$ is increased, the allowed maximum value of $\#B\_TSV$

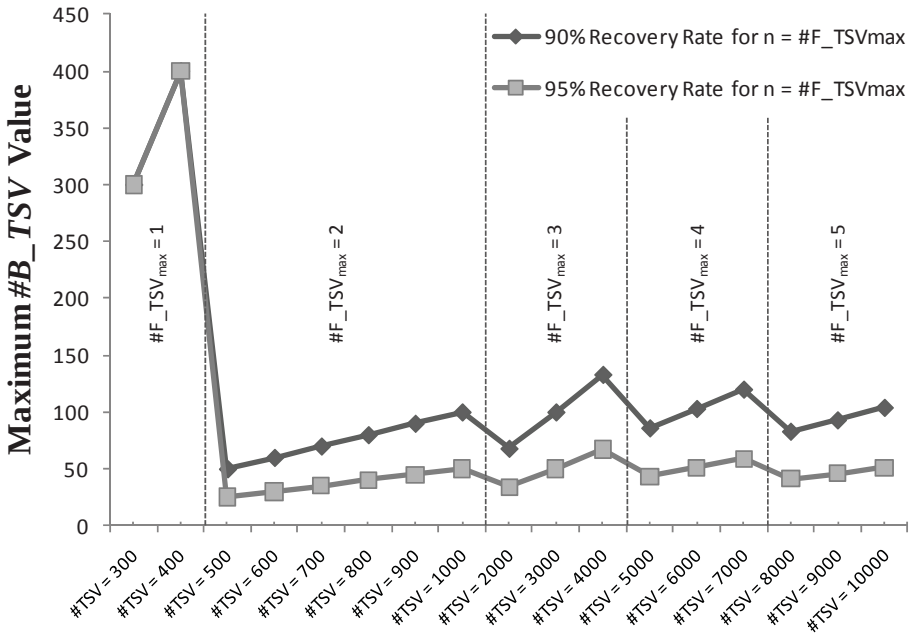



Figure 4.10: Limits on $\#B\_TSV$

drops dramatically. In the worst case ($\#TSV_{tier} = 500$), to achieve 90% and 95% recovery rates, the allowed maximum values of $\#B\_TSV$ are 25 and 50, respectively.

A further analysis is to compute the overall yield. For given value of $\#TSV_{tier}$ and the maximum value of $\#B\_TSV$, the recovery rate for $n$ can be computed. Let $R\_Rate_{(\#TSV_{tier},n,\#B\_TSV)}$ stand for the recovery rate for given $\#TSV_{tier}$, $n$ and $\#B\_TSV$. The overall yield can be computed as Equation (4.3).

$$Yield_{(\#TSV_{tier},\ n,\ \#B\_TSV)} = P_{f\_tsv=0} + \sum_{n=1}^{\#TSV_{tier}} P_{f\_tsv=n} \times R\_Rate_{(\#TSV_{tier},\ n,\ \#B\_TSV)} \quad (4.3)$$

Since the value of $P_{f\_tsv=n}$ is very small when $n > \#F\_TSV_{max}$, we can

rewrite the function as Equation (4.4).

$$Yield_{(\#TSV_{tier},\ n,\ \#B\_TSV)} = P_{f\_tsv=0} + \sum_{n=1}^{\#F\_TSV_{max}} P_{f\_tsv=n} \times R\_Rate_{(\#TSV_{tier},\ n,\ \#B\_TSV)} \quad (4.4)$$

For example, when $\#TSV_{tier} = 500$, the value of $\#F\_TSV_{max}$ is 2. Using Equation (4.1), the values of $P_{f\_tsv=0}$, $P_{f\_tsv=1}$, and $P_{f\_tsv=2}$ can be computed as 95.1227%, 4.7566%, and 0.1187%, respectively. The recovery rate for $n = 1$ is 100% based on our proposed architecture. Let the recovery rate for $n = 2$ be set to 90%. The overall yield can be computed as

$$P_{f\_tsv=0} + P_{f\_tsv=1} \times 100\% + P_{f\_tsv=2} \times 90\% = 99.98613\%.$$

A complete analysis on the overall yields for $\#TSV_{tier} = \{300\sim10000\}$ is presented in Figure 4.11. Similar to Figure 4.10, regions partitioned by dash lines denote the increase in $\#F\_TSV_{max}$. In all cases, the yields are higher than 99.4%. Another observation is that, increasing the recovery rate for $n = \#F\_TSV_{max}$ from 90% to 95% only results in limited improvement on the overall yield. Therefore, setting the recovery rate for $n = \#F\_TSV_{max}$ to 90% is sufficient in most cases. Considering the analysis results in Figures 4.10 and 4.11, limiting the number of TSVs in each TSV block to be no greater than 50 results in yields higher than 99.4%.

## 4.5 Comparison with Previous Work

### 4.5.1 Experiment Setup

To understand the overheads of different redundant TSV schemes, a pseudo TSV model needs to be assumed so that the timing and power behaviors of

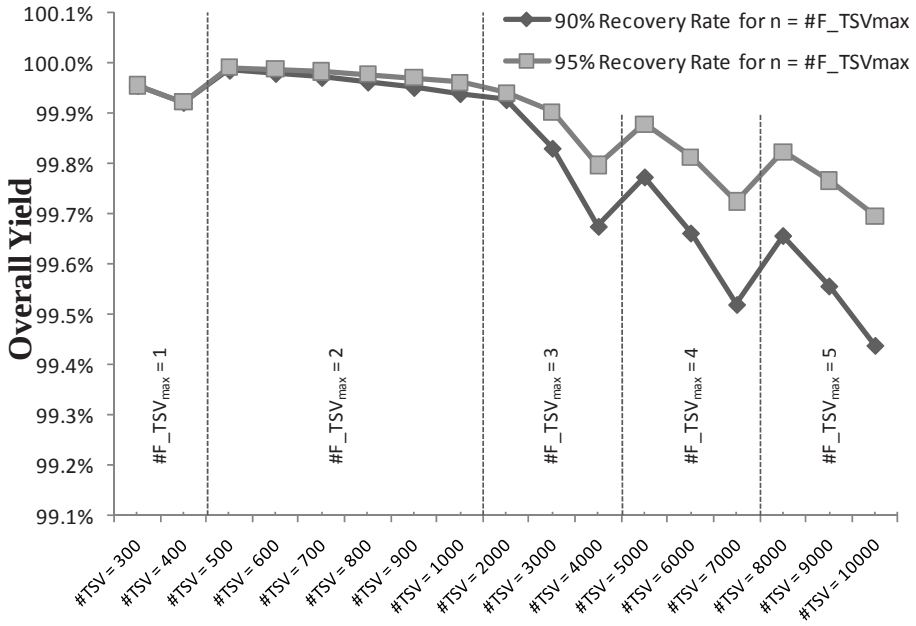



Figure 4.11: The Overall Yields

each redundant scheme can be evaluated. In our evaluation, the parameters of TSV are defined based on via-first/via-middle TSV processes with face-to-back bonding technology. The parameters of the TSV used in our evaluation are summarized in Table 4.2.

Currently, copper is still the most widely used material for metallization

Table 4.2: TSV Parameters in Our Evaluation

| **Parameter** | **Value** |
|---|---|
| Material for Metallization | Copper ($Cu$) |
| TSV Diameter ($d$) | 8$\mu$m |
| TSV Height ($h$) | 50$\mu$m |
| Material of Dielectric Layer | Silicon Dioxide ($SiO_2$) |
| Thickness of Dielectric Layer ($t_{ox}$) | 0.15$\mu$m |

due to its mature process technology. Thus, copper is selected as the metallization material for TSV in our assumption. For via-first or via-middle processes, the diameter, $d$, of a TSV ranges from 3$\mu$m to 20$\mu$m [75]$\sim$ [78]. In our evaluation, the value of $d$ is assumed to be 8$\mu$m. The height, $h$, of a TSV is identical to wafer thickness after thinning process. Presently, the value of $h$ is less than 100$\mu$m in modern 3D IC processes [48]. Modern technology allows the value of $h$ to be decreased to 30$\mu$m [75] [78]. In our assumption, the value of $h$ is set to 50$\mu$m. Silicon dioxide is selected as the dielectric material for TSVs. The sidewall thickness, $t_{ox}$, is reported to range from 0.1$\mu$m to 1$\mu$m [75] [79] [80]. For TSVs with smaller diameters, it is technically difficult to obtain a larger value of $t_{ox}$ due to the process technology of deposition [81]. In our assumption, the value of $t_{ox}$ is set to 0.15$\mu$m.

Since TSVs are very large as compared with metal wires, the resistance of TSVs can be ignored. The capacitance of a TSV is determined by the dimension and the property of dielectric material. According to the parameters in Table 4.2, the capacitance of a TSV can be computed using Equation (4.5) where $\varepsilon_r$ and $\varepsilon_0$ are the relative dielectric constant of silicon dioxide and the permittivity of vacuum. The radius of TSV, $r_{tsv}$, can be computed as $d/2$.

$$C_{tsv} = \frac{2\pi\varepsilon_r\varepsilon_0 h}{ln((r_{tsv} + t_{ox})/r_{tsv})} \tag{4.5}$$

The value of $\varepsilon_r$ is dependent on process technology. In modern industry, the value of $\varepsilon_r$ ranges from 3.5 $\sim$ 3.9 [82] [83]. In our evaluation, the value of $\varepsilon_r$ is assumed to be 3.7. The capacitance of a TSV can therefore be computed as 278fF. The loading capacitance caused by bond pad and soldering material is very small ($<$ 1fF) and can be neglected [84].

For simplicity, we refer the TSV structure where each vertical interconnect is realized by one TSV as the *single TSV* structure. The *single TSV* structure is used in the redundant TSV scheme proposed in this chapter and the *switching box* method. For the *twins* structure proposed by HRI, each vertical interconnect is realized by a pair of TSVs [65]. Therefore, the loading capacitance of the *twins* structure is doubled.

Since the loading capacitance of TSVs is very large as compared with the loading capacitance of 2D wires, buffers are required to drive TSVs. In our evaluation, buffers of different sizes (2X∼16X) in 90nm CMOS technology are tested. Assume that the input transition time is set to 0.1ns. The relation between output loading capacitance and output transition time is shown in Figure 4.12. Two dash lines denoted as $C_{Single}$ and $C_{Twins}$ in Figure 4.12 represent the loading capacitance of the *single TSV* and the *twins* structures. Let the loading capacitance of the wires connected to the TSV, and the input pins connected to these wires be denoted as $C_{iLoad}$. For $C_{Single}$ and $C_{Twins}$, two dotted lines for $C_{iLoad}$ = 15fF and $C_{iLoad}$ = 50fF are drawn. Let $T_{max\_trans}$ stand for the expected maximum transition time and $T_{max\_trans}$ be set to 0.1ns, which is the same as the input transition time. According to Figure 4.12, when $C_{iLoad}$ is small (no greater than 15fF), the minimum buffer sizes required by the *single TSV* and *twins TSV* structures are 6X and 12X, respectively. When $C_{iLoad}$ is greater than 50fF, the minimum buffer sizes required by the *single TSV* and *twins TSV* structures are increased to 8X and 16X, respectively. In general, to obtain the same output transition time, the buffer size required by the *twins* structure is 1.5∼2 times larger than that required by the *single TSV* structure. The buffers required for driving TSVs

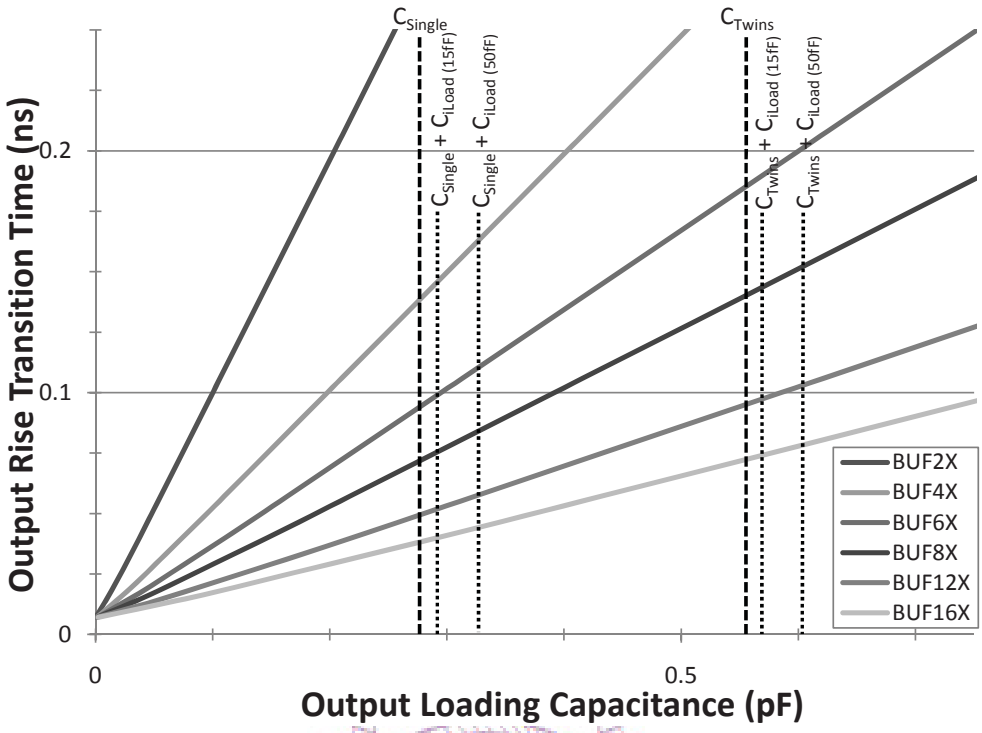



Figure 4.12: Relation between Output Loading Capacitance and Rise Transition Time

lead to overheads of delay, area, and power consumption.

To allow *switching box* method proposed in 3D DRAM [63] to be used in general designs, additional testing circuits are required to test the connectivity of TSVs. In our evaluation, the testing circuits proposed in Section 4.3.2 are added to the *switching box* design. 3-1 MUXs and 1-3 DEMUXs are used to construct the *switching box* circuits as shown in Figure 4.13.

### 4.5.2 Comparisons of These Methods

In Section 4.5.1, the sizes of the buffers that are required by the *single TSV* and the *twins* structures have been discussed. Since large buffers which are

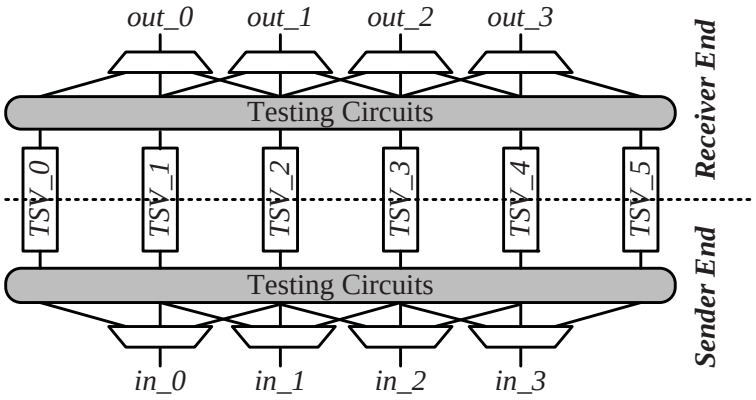



Figure 4.13: *Switching Box* Implementation in Our Evaluation

used to drive TSVs have large input capacitance, small buffers are used to drive the large buffers. In the following analysis, we assume that a 1X buffer is inserted to drive the large buffer for each TSV. Therefore, when $C_{iLoad}$ is no greater than 15fF, *1X + 6X* and *1X + 12X* buffer chains are used for the *single TSV* and the *twins* structures, respectively. When $C_{iLoad}$ is between 15fF to 100fF, the *1X + 8X* and *1X + 16X* buffer chains are used. The circuits required in the *switching box* and our method are constructed using a 90nm CMOS cell library. To accurately evaluate the area overhead of our proposed redundant TSV scheme, in each TSV block, the number of TSVs and flip-flops in the testing circuits on the sender end are assumed to be 50 and 10, respectively. According to these assumptions, the overheads of one TSV in terms of timing, leakage power, and area in different schemes are summarized in Table 4.3.

In Table 4.3, rows labeled *Original* refer to TSV structure with no redundancy. In these two rows, only the buffers used to drive the *single TSV* structure are considered. The delay caused by buffers and additional control

Table 4.3: Overheads of Different TSV Structures in Terms of Delay, Power, and Area (per TSV)

| **TSV Structure** | **$C_{iLoad}$ (fF)** | **Timing Delay (ns)** | **Leakage Power (pW)** | **Area (NAND2 Gate Count)** | | |
|---|---|---|---|---|---|---|
| | | | | *Circuits* | *TSV Body* | |
| | | | | | *90nm* | *32 ∼ 45nm* |
| *Original (No Redundancy)* | ≤ 15 | 0.076792 | 96102.7 | 4.25 | 16 | 100 |
| | 15 ∼ 100 | 0.080204 | 124962.2 | 5.25 | | |
| *Twins Structure* | ≤ 15 | 0.087096 | 186281.2 | 7.25 | 32 | 200 |
| | 15 ∼ 100 | 0.093229 | 240402.2 | 9.25 | | |
| *Switching Box Method* | ≤ 15 | 0.239534 | 190106.8 | 32.43 | 24 | 150 |
| | 15 ∼ 100 | 0.242508 | 220936.5 | 33.43 | | |
| *Our Method* | ≤ 15 | 0.194872 | 161255.2 | 28.35 | 16.32 | 102 |
| | 15 ∼ 100 | 0.196374 | 186311.4 | 29.35 | | |

circuits in each redundant TSV scheme is listed in column *Timing Delay*. In column *Leakage Power*, the leakage power of active components in normal mode is computed. The leakage power of testing circuits is ignored because these circuits are usually power gated in normal mode. The area of different TSV structures is summarized in the columns labeled *Area* expressed as the numbers of NAND2 gate count. To fairly compare the area overhead of different TSV structures, the average area overhead for one signal TSV is computed. Take our proposed method as an example. Let the area for a TSV block of 50 TSVs be computed where there are 49 signal TSVs and 1 redundant TSV. Then, the area overhead is divided by 49.

The second is the die area occupied by the physical bodies of TSVs. This area is reported in the columns labeled *TSV Body*. The area overhead of TSV body is analyzed as follows. For a TSV of diameter, $d\mu$m, the die area occupied by a TSV is at least $d^2\mu m^2$. In the 90nm CMOS technology, the size of a 1X NAND gate is about $4\mu m^2$. Therefore, the area occupied by one TSV body is equivalent to $\frac{d^2}{4}$ or more NAND gates. In more advanced

process technology, the size of a logic gate is greatly reduced. However, the size of a TSV is not likely to be reduced due to TSV fabrication. According to ITRS 2009 edition [73], under 32nm or 45nm process technology, the size of a 1X NAND gate may ranges from $0.5\mu\text{m}^2 \sim 1\mu\text{m}^2$. This means, the die area occupied by one TSV may be equivalent to $d^2 \sim 2d^2$ NAND gates. For example, when $d = 8\mu$m, each TSV is equivalent to $64 \sim 128$ NAND gates. In Table 4.3, we assume that $d = 8\mu$m and each TSV is equivalent to 16 NAND gates in 90nm process technology and 100 NAND gates in 32nm $\sim$ 45nm process technology. Based on these assumptions, the area overhead of TSV body for one signal TSV is computed as

$$TSV_{area} \times \frac{\#TSV_{all}}{\#TSV_{signal}}.$$

This area is reported in the columns labeled *TSV Body*.

In addition to the die area occupied by the bodies of TSVs, the chip size of each tier may needs to be increased due to the pitch constraint of TSVs [48]. In Table 4.3, pitch constraint of TSV is not considered. Through this analysis, we should understand that the area overhead caused by TSV can be very large and may dominate the area of each TSV structure.

According to Table 4.3, we can see that our redundant TSV scheme leads to minimum area overhead as compared with the *twins* stricture and the *switching box* method. As for the overhead of timing delay, our scheme results in smaller delay as compared with the *switching box* method. However, as compared with the *twins* structure, the timing delay of our proposed method is larger. In the *twins* structure, only the delay of buffers are induced. Although our proposed method leads to smaller buffers for TSVs, additional

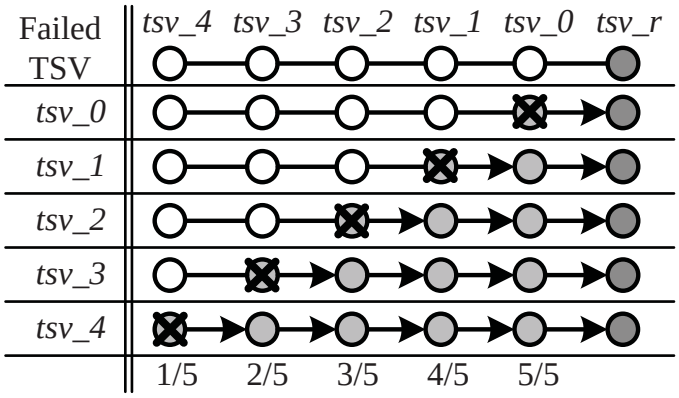



Figure 4.14: All Possible Shifting Situations for a *TSV-chain* of Size 6 when 1 TSV is Failed

MUX and DEMUX gates on signal paths results in larger path delay. In terms of leakage power, our proposed method outperforms the *twins* structure and the *switching box* method. Although the *switching box* method results in larger delay and leakage power as compared with the other two methods, this method provides an unique property where the delays of all signals are almost identical. This property is desired for special applications.

## 4.6 Design Flow and *TSV-Chain* Design

### 4.6.1 Design Issues for Timing

When a TSV is failed, according to the position of the failed TSV in a *TSV-chain*, one or more signals need to be shifted. Due to the chaining structure, even under the assumption that each TSV has identical failure rate, the probability for each TSV to be shifted varies. Figure 4.14 shows this situation.

Assume that 1 TSV is failed in a *TSV-chain* of size 6, all possible shifting

situations are enumerated in Figure 4.14. When no TSV is failed, no shifting is required. The *TSV-chain* is shown in the first row where the redundant TSV is denoted as *tsv_r* at the rightmost column. For each row below, the leftmost column indicates which TSV is failed and the right columns show the shifting situation. The last row lists the shifting probabilities of the TSVs in the *TSV-chain* when 1 TSV is failed. For *tsv_0*, no matter which TSV in the *TSV-chain* is failed, it is always shifted because it is on the position next to the redundant TSV. On the contrary, *tsv_4*, which is at the head position of the *TSV-chain*, need not be shifted unless *tsv_4* itself is failed. In terms of extra delays introduced by signal shifting, this property of *TSV-chain* indicates that the probability that the delay of a signal linked by a TSV is increased depends on the position of the TSV in the *TSV-chain*. This means, for signals that are timing critical, it is preferable to assign these signals at the head parts of *TSV-chains*.

An evaluation for an extreme case where only one signal is timing critical is shown in Figure 4.15. The x-axis stands for the number of TSVs in a *TSV-chain* and the y-axis stands for the probability that the timing critical signal is shifted. The line denoted as "Unaware" represents that the timing critical signal has equal probability to be located at any position of a *TSV-chain*. And the line denoted as "Timing Aware" represents that the timing critical signal is always located at the beginning of a *TSV-chain*. Assume that the failure rate of each TSV is identical and there is only one failed TSV. The result in Figure 4.15 shows that, in "Unaware" cases, the probabilities for the timing critical signal to be shifted are greater than 50% in all cases. On the contrary, by assigning the timing critical signal to the head of a *TSV-chain*,

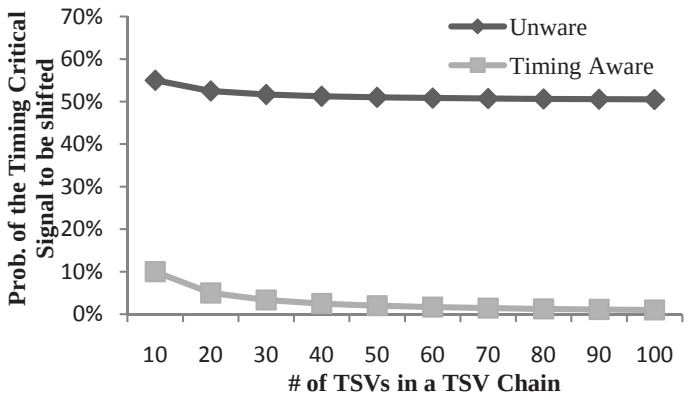



Figure 4.15: Evaluation on the Possibility for the Timing Sensitive Signal to Be Shifted

the probability is reduced to 2.93% in average. Based on the evaluation, **timing sensitive signals should always be routed through the TSVs located at the head of *TSV-chains***. This is one of the guideline that should be followed when designing *TSV-chains*.

The next issue is to minimize the delay caused by signal shifting. This can be done by minimizing the distance between the connected TSVs in a *TSV-chain*. Since TSVs in each block are placed in a grid-based structure, by requiring the connected TSVs in a *TSV-chain* to be neighbors in the grid-based structure, minimal and fixed shifting delay can be guaranteed. This also makes the shifting delay predictable in early design stages. Thus, the second guideline for *TSV-chain* design is that **any two connected TSVs in a *TSV-chain* must be next to each other in the grid-based structure**.

### 4.6.2 *TSV-chain* Design Problem

For each TSV block in a plane, the structure of the *TSV-chain* needs to be considered. The analysis in Section 4.6.1 indicates that timing critical signals should always be routed through the TSVs located at the head parts of *TSV-chains*. In current design flow, signals that are assigned to each TSV block are roughly determined in floorplan stage. However, the exact assignment of signals to TSVs is not necessarily to be done in this stage. From the perspective of physical design, leaving the assignment of signals to TSVs to be done in routing stage is beneficial to minimize wire length. Therefore, in addition to the guidelines obtained in Section 4.6.1, the design of *TSV-chain* should also consider routing issues.

Based on the concept of bounding box, discussion on wire length is given first. For two pins on two different tiers to be connected, the relation between the bounding box of these two pins and a TSV block can be listed as follows. First, the bounding box and the TSV block can be non-overlapped. In this situation, only going through a TSV on the boundary of the TSV block can result in minimum wire length. Next, the TSV block can be either partially or completely overlapped by the bounding box. In this situation, any TSV that is overlapped by the bounding box can result in minimum wire length. Unless the bounding box is completely contained in the TSV block, a TSV on the boundary of the TSV block can always be found for minimum wire length. The discussion shows that, TSVs on the boundary of a TSV block have higher probabilities to be routed through for minimum wire length. These TSVs should be assigned as head parts of *TSV-chains*.

A spiral-style chaining policy is proposed for *TSV-chain* design. In a

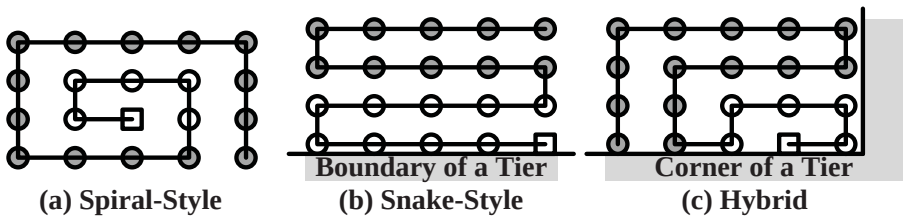



Figure 4.16: *Chaining Styles*

TSV block, by picking a TSV in the central position to be the starting point, spiral-style chaining results in a routing path where all TSVs on the boundary are at one end. The starting TSV is assigned as redundant TSV while the other end becomes the head of a *TSV-chain*. An example for a $4 \times 5$ TSV block is shown in Figure 4.16(a) where TSVs in grey are head and good for timing critical signals. In routing stage, routers can choose to assign timing-critical signals to TSVs that are on the boundary of a TSV block. This can reduce the probability for a timing critical signal to be shifted.

The spiral-style chaining policy is appropriate for a TSV block that is not on the boundary or the corner of a tier. For a TSV block located on the boundary of a tier, most signals assigned to that TSV block are connected from the opposite side of the tier boundary. In this case, a snake-style chaining policy satisfies the requirement. The result is shown in Figure 4.16(b). For a TSV block located on the corner, most signals assigned to that TSV block are connected from the counter direction of the tier corner. In this case, a hybrid chaining policy as shown in Figure 4.16(c) becomes the best candidate. Based on the position of each TSV block, one of these three chaining policies is applied.

### 4.6.3 Physical Design Flow Considering *TSV-Chain*

In current design flow for 3D ICs , 3D partitioning first takes place to determine in which tier each design blocks to be placed. The number of required TSVs for signals between two consecutive tiers is determined in this stage. Next, in floorplan stage, blocks with fix area but unknown dimensions are placed in each tier. To provide communication links between blocks in different tiers, TSV blocks are placed. The number of signals to be assigned to each TSV block as well as the position of each TSV block are roughly determined in this stage. Based on the discussion in Section 4.4, the number of TSVs in each TSV block should be limited. Partitioning may be required for large TSV blocks. Based on the position of each TSV block, the structure of each *TSV-chain* is determined. In place and route stage, routers should be aware of the *TSV-chain* structure in each TSV block. Based on design constraints and requirements, router assigns timing critical signals to TSVs that are located at the head of *TSV-chains*. The overall design flow for *TSV-chain* is shown in Figure 4.17.

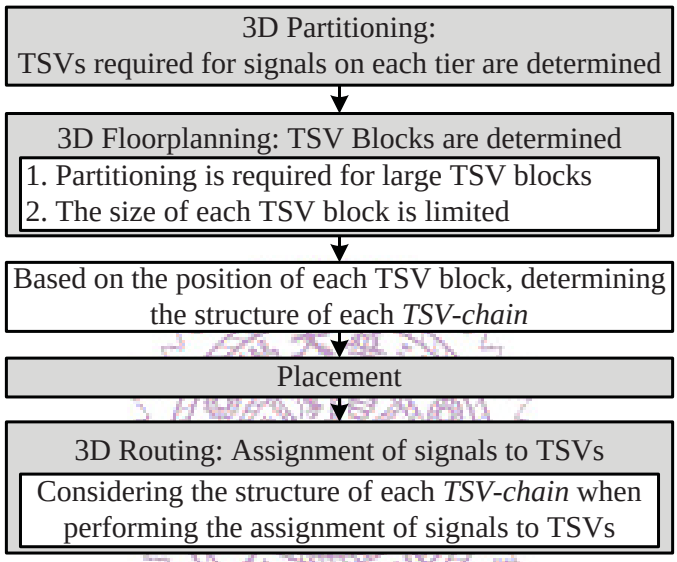



Figure 4.17: Proposed Design Flow for *TSV-Chain*

# Chapter 5

# Conclusion

In this dissertation, architecture designs and optimization techniques have been proposed for modern integration technologies in cache, memory and interconnect levels.

In cache level, we have found that a fixed expansion scheme supported by original *expandable cache* is insufficient for applications that are complex and irregular, A refined cache structure of *expandable cache* which considers programs behavior and has many flexible expansion schemes has been proposed. By adding an extra register and a small number of XOR gates to the cache design, the expansion scheme of our design can be dynamically changed by executing configuration instructions which are inserted at compile time. This new design leads to lower miss rate and better power efficiency. The experimental results of *SPEC CPU2000* have shown that our proposed cache design effectively improves the miss rate by 14.74% as compared with the original *expandable cache*. In terms of energy improvement ratio, our method is 5.62% higher than that of *expandable cache* when the baseline is set as the energy consumption of 2-way set-associative.

For 3D memory systems, we have proposed a static thermal management scheme for stacked DRAM dies. 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. For single-core systems, experiments show that our method can reduce temperature of memory system by 17.1°C as compared to a straightforward mapping in the best case, and 13.3°C in average. For systems with 4 cores, the temperature reductions are 9.9°C and 11.6°C in average when L1 cache of each core is set to 4KB and 8KB, respectively.

For the communication links for dies in vertical direction, a new redundant TSV architecture with reasonable cost for ASICs has been proposed. Our proposed redundant TSV scheme is scalable and can be adjusted to fit the failure rate of TSV for different TSV processes and bonding technologies. Testing circuits for the proposed architecture have been introduced. Design issues including recovery rate and timing problem have been investigated. Based on probabilistic models, the new design can successfully recover most of the failed TSVs and increase the yield to 99.4%. This can effectively reduce the cost of manufacturing 3D designs.

# Bibliography

[1] L. Lee, S. Kannan, and J. Fridman, "MPEG4 Video Codec on a Wireless Handset Baseband Syatem," *In Proc. Workshop Media and Signal Processors fir Embedded Systems and SOCs,* 2004.

[2] G. Bournoutian and A. Orailoglu,"Miss Reduction in Embedded Processors Through Dynamic, Power-Friendly Cache Design" in *Proc. Design Automation Conference (DAC),* June, 2008, pp. 304-309.

[3] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad Memory: A Design Alternative for Cache On-Chip Memory in Embedded Systems," in *Proc. 10th International Symposium on Hardware/Software Codesign,* New York, 2002.

[4] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy efficient memory structure," in *Proc. 30th International Symposiun on Microarchitecture,* 1997, pp. 184-193.

[5] A. Janapsatya, S. Parameswaran, and A. Ignjatovic, "Hardware/Software Managed Scratchpad Memory for Embedded System," in *Proc. International Conference on Computer-Aided Design (ICCAD),* 2004.

[6] C. T. Wu, A.-C. Hsieh, and TingTing Hwang, "Instruction Buffering for Nested Loops in Low-Power Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol. 14, No.7, July 2006, pp. 780-784.

[7] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel, "Assigning Program and Data Objects to Scratchpad for Energy Reduction," in *Proc. Design, Automation and Test in Europe (DATE),* 2002.

[8] M. Verma, L. Wehmeyer, and Peter Marwedel, "Cache-Aware Scratchpad Allocation Algorithm," in *Proc. Design, Automation and Test in Europe (DATE),* 2004.

[9] F. Angiolini, L. Benini, and A. Caprara, "An Efficient Profile-based Algorithm for Scratchpad Memory Partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 24, No. 11, Nov. 2005, pp. 1660-1676.

[10] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and J. M. Mendias, "An Integrated Hardware/Software Approach for Run-Time Scratchpad Management," in *Proc. Design Automation Conference (DAC)*, 2004.

[11] S. Udayakumaran, A. Dominguez, and Rajeev Barua, "Dynamic Allocation for Scratch-Pad Memory using Compile-Time Decisions," *ACM Transactions on Embedded Computing Systems (TECS),* Vol. 5, Issue 2, May 2006.

[12] D. H. Albonesi, "Selective Cache Ways: On-demand Cache Resource Allocation," *32th International Symposium on Microarchitecture*, 1999, pp. 248-259.

[13] A. González, C. Aliagas, and M. Valero, "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality," *9th International Conference on Supercomputing*, 1995, pp. 338-347.

[14] P. Petrov and A. Orailoglu, "Performance and Power Effectiveness in Embedded Processors - Customizable Partitioned Caches," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2001, pp. 1309-1318.

[15] N. P. Jouppi, "Improving Direct-mapped Cache Performance by the Addition of Small Fully-Associatve Cache and Prefetch Buffers," *SIGARCH Computer Architecture News*, 1990, pp. 364-373.

[16] A. Agarwal and S. D. Pudar, "Column-Associative Caches: a Techniques for Reducing the Miss Rate of Direct-Mapped Caches," *SIGARCHI Computer Architecture News*, 1993, pp. 179-190.

[17] ARM926 Processor, http://www.arm.com/products/processors/classic/arm9/

[18] H. Homayoun, S. Pasricha, M. Makhzan, and A. Veidenbaum, "Dynamic Register File Resizing and Frequency Scaling to Improve Embedded Processor Performance and Energy-Delay Efficiency," *Design Automation Conference*, June, 2008, pp. 68-71.

[19] F. Catthoor, K. Danckaert, K. K. Kulkarni, E. Brockmeyer, P. G. Kjeldsberg, T. van Achteren, and T. Omnes, "Data Access and Storage Management for Embedded Programmable Processors," *Springer*, ISBN: 978-0-7923-7689-7, 2002.

[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] SPEC CPU2000 Benchmarks, http://www.spec.org/cpu/.

[22] S. J. Wilton, N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal on Solid-State Circuits*, 31(5): pp. 677-688, 1996.

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

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

[25] 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*, 1995, pp 466-471.

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

[27] M. X. Wang, K. Suzuki, W. Dai, Yee L. Low, K. J. O'conner 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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

[39] *HY5DU12822D(L)TP-xI/HY5DU121622D(L)TP-xI 512Mb DDR SDRAM Technical Data Sheet*, http://www.hynix.com/

[40] 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

[41] 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

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

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

[44] K. Banerjee, S. Souri, P. Kapur, and K. Saraswat, "3D ICs: A Novel Chip Design for Improving Deep Submicron Interconnecct Performance and System-on-Chip Integration," in *Proc. IEEE,* vol. 89, no. 5, 2001, pp. 602-633.

[45] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon, "Demistifying 3D ICs: The Pros and Cons of Going Vertical," *IEEE Design & Test of Computers,* vol. 22, no. 6, Nov./Dec., 2005, pp. 498-510.

[46] J. Burns, L. McIlrath, C. Keast, C. Lewis, A. Loomis, K. Warner, and P. Wyatt, "Three-Dimensional Integrated Circuit for Low Power, High-Bandwidth Systems on a Chip," *ISSCC Dig. of Tech. Papers,* Feb., 2001, pp. 268-269.

[47] P. R. Morrow, M. J. Kobrinsky, S. Ramanathan, C.-M. Park, M. Harmes, V. Ramachandrarao, H.-M. Park, G. Kloster, S. List, and S. Kim, "Wafer-Level 3D Interconnects via Cu Bonding," in *Proc. AMC,* 2004, pp. 125-130.

[48] P. Garrou, C. Bower, and P. Ramm, "Handbook of 3D Integration: Technology and Application of 3D Integrated Circuits," *Weinheim: WILEY-VCH Verlag GmbH & Co. KGaA,* 2008, vol. 1-2.

[49] J. Cong, J. Wei, and Y. Zhang, "A Thermal-Driven Floorplanning Algorithm for 3D ICs," in *Proc. ICCAD,* Nov. 2004, pp. 306-313.

[50] W.-L. Hung, G. M. Link, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Interconnect and Thermal-Aware Floorplanning for 3D Microprocessors," in *Proc. ISQED,* Mar., 2006, pp. 104-109.

[51] P. Zhou, Y. Ma, Z. Li, R. P. Dick, L. Shang, H. Zhou, X. Hong, and Q. Zhou, "3D-STAF: Scalable Temperature and Leakage Aware Floorplanning for Three-Dimensional Integrated Circuits," in *Proc. ICCAD,*, Nov. 2007, pp. 590-597.

[52] M.-C. Tsai, T.-C. Wang, and T. T Hwang, "Through-Silicon Via Planning in 3D Floorplanning," *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, to be published.

[53] B. Goplen and S. Sapatnekar, "Efficient Thermal Placement of Standard Cells in 3D ICs using a Force Directed Approach," in *Proc. ICCAD,* Nov. 2003, pp. 86-89.

[54] J. Cong, G. Luo, J. Wei, and Y. Zhang, "Thermal-Aware 3D IC Placement Via Transformation," in *Proc. ASP-DAC,* Jan. 2007, pp. 780-785.

[55] B. Goplen and S. Sapatnekar, "Placement of Thermal Vias in 3-D ICs Using Various Thermal Objectives," *IEEE Trans. Comput.-Aided Des, Integr. Syst.,* vol. 25, no. 4, Apr. 2006, pp. 692-709.

[56] H. Yu, Y. Shi, L. He, and T. Karnik, "Thermal Via Allocation for 3D ICs Considering Temporally and Spatially Variant Thermal Power", *IEEE TVLSI,* vol. 16, no. 12, December 2008, pp. 1609-1619.

[57] T. Zhang, T. Zhan, and S. Sapatnekar, "Thermal-Aware Routing in 3D ICs", *in Proc. ASP-DAC,* March 2006, pp. 309-314.

[58] Hao Yu, Joanna Ho, and Lei He, "Allocating Power Ground Vias in 3D ICs for Simultaneous Power and Thermal Integrity," *ACM TODAES,* vol. 14, no. 3, May 2009, pp. 41-71.

[59] M. Pathak, S. K. Lim, "Performance and Thermal-Aware Steiner Routing for 3-D Stacked ICs," *IEEE TCAD,* vol. 28, no. 9 Sep. 2009, pp. 1373-1386.

[60] P. Zhou, K. Sridharan, and S. Sapatnekar, "Optimizing Decoupling Capacitors in 3D Circuits for Power Grid Integrity," IEEE Des. Test Comput., vol. 26, no. 5, Sep./Oct. 2009, pp. 15-25.

[61] P. Falkenstern, Y. Xie, Y.-W. Chang, and Y. Wang, "Three-Dimensional Integrated Circuits (3D IC) Floorplan and Power/Ground Network Co-Synthesis," in *Proc. ASP-DAC,* pp. 169-174, Jan. 2010.

[62] H.-T. Chen, H.-L. Lin, Z.-C. Wang and T. T. Hwang, "New Architecture for Power Network in 3D IC," presented in *Proc. DATE,* Grenoble, France, Mar. 2011.

[63] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J. Ahn, D. Kwon, J. H. Kim, J.-W. Lee, H.-S. Joo, W.-S. Kim, H.-K. Kim, E.-M. Lee, S.-R. Kim, K.-H. Ma, D.-H. Jang, N.-S. Kim, M.-S. Choi, S.-J. Oh, J.-B. Lee, T.-K. Jung, J.-H. Yoo, and C. Kim, "8Gb 3D DDR3 DRAM Using Through-Silicon-Via Technology," *ISSCC Dig. of Tech. Papers,* Feb., 2009, pp. 130-131.

[64] I. Loi, S. Mitra, T. H. Lee, S. Fujita, and L. Benini, "A Low-Overhead Faule Tolerance Scheme for TSV-Based 3D Network on Chip Links," in *Proc. ICCAD,* 2008, pp. 598-602.

[65] N. Miyakawa, T. Maebashi, N. Nakamura, S. Nakayama, E. Hashimoto, and S. Toyoda, "New Multi-Layer Stacking Technology and Trial Manufacture," Honda Research Institute Japan Co. Ltd., Nov. 2007.

[66] H.-H. Lee and K. Chakrabarty, "Test Challenges for 3D Integrated Circuits," *IEEE Design & Test of Computers,* vol. 26, no. 5, Sep./Oct., 2009, pp. 26-35.

[67] R. Patti, "Three-Dimensional Integrated Circuits and the Future of System-on-Chip Designs," *Proc. IEEE*, vol. 84, no. 6, June 2006, pp. 1214-1224.

[68] A. W. Topol, D. C. La Tulipe, L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and

M. Ieong, "Three Dimensional Integrated Circuits," *IBM Journal of Research and Development,* vol. 50, no. 4/5, July/Sepetember 2006, pp. 491-506.

[69] R. Patti, "Impact of Wafer-Level 3D Stacking on the Yield of ICs," *Future Lab Intl.,* issue 23, July 2007, pp. 103-105.

[70] N. Miyakawa, "A 3D Prototyping Chip based on a Wafer-Level Stacking Technology," *Proc. ASP-DAC,* Jan. 2009, pp. 416-420.

[71] B. Swinnen, W. Ruythooren, P. De Moor, L. Bogaerts, L. Carbonell, K. De Munck, B. Eyckens, S. Stoukatch, D. Sabuncuoglu Tezcan, Z. Tokei, J. Vaes, J. Van Aelst, and E. Beyne, "3D Integration by Cu-Cu Thermo-Compression Bonding of Extremely Thinned Bulk-Si Die Containing 10 $\mu$m Pitch Through-Si Vias," in *Proc. IEDM,* Dec. 2006.

[72] A. W. Topol, D. C. La Tulipe, L. Shi, S. M. Alam, D. J. Frank, S. E. Steen, J. Vichiconti, D. Posillico, M. Cobb, S. Medd, J. Patel, S. Goma, D. DiMilia, M. T. Robson, E. Duch, M. Farinelli, C. Wang, R. A. Conti, D. M. Canaperi, L. Deligianni, A. Kumar, K. T. Kwietniak, C. DEmic, J. Ott, A. M. Young , K. W. Guarini, and M. Ieong, "Enabling SOI-Based Assembly Technology for Three-Dimensional Integrated Circuits," in *Proc. IEDM*, Dec. 2005, pp. 352-355.

[73] ITRS, "International Technology Roadmap for Semiconductors," 2009. [Online]. Available: http://www.itrs.net/

[74] A. M. Rodin, J. Callaghan, and N. Brennan, "High Throughput Low CoO Industrial Laser Drilling Tool", 2008. [Online]. Available: http://www.emc3d.org/

[75] L. W. Schaper, S. L. Burkett, S. Spiesshoefer, G. V. Vangara, Z. Rahman, and S. Polamreddy, "Architectural Implications and Process Development of 3-D VLSI Z-Axis Interconnects Using Through Silicon Vias," *IEEE Trans. Adv. Packag.,* vol. 28, no. 3, Aug. 2005, pp. 356-366.

[76] A. Shayan, X. Hu, H. Peng, C.-K. Cheng, W. Yu, M. Popovich, T. Toms, and X. Chen, "Reliability Aware Through Silicon Via Planning for 3D Stacked ICs," in *Proc. DATE),* Apr. 2009, pp. 288-291.

[77] M. Puech, J. M. Thevenoud , J. M. Gruffat, N. Launay, N. Arnal, and P. Godinat, "DRIE achievements for TSV covering Via First and Via Last Strategies," *ALCATEL Micro Machine System,* Annecy, France, 2008. [Online]. Available: http://www.alcatelmicromachining.com/.

[78] Sung Kyu Lim, "TSV-Aware 3D Physical Design Tool Needs for Faster Mainstream Acceptance of 3D ICs," presented at the *47th ACM Design Autom. Conf. Knowledge Center* Anaheim, CA., 2010/

[79] N. Ranganathan, L. Ebin, L. Linn, W. S. V. Lee, O. K. Navas, V. Kripesh, and N. Balasubramanian, "Integration of High Aspect Ratio Tapered Silicon Via for Silicon Carrier Fabrication," *IEEE Trans. Adv. Packag.,* vol. 32, issue 1, Feb. 2009, pp. 62-71.

[80] J. S. Pak, C. Ryu, and J. Kim, "Electrical Characterization of Trough Silicon Via (TSV) Depending on Structural and Material Parameters

based on 3D Full Wave Simulation," presented at the *EMAP,* Daejeon, South Korea, Nov. 2007.

[81] X.-P. Wang and W.-Y. Yin, "Multi-Physics Characterization of Through Silicon Vias (TSV) in the Rresence of a Periodic EMP," presented at the *EDAPS*, Shatin, Honk Kong, Dec. 2009.

[82] J. Plummer and P. Griffin, "Material and process limits in silicon VLSI technology," in *Proc. IEEE,* vol. 89, issue. 3, Mar. 2001, pp.240-258.

[83] D. Sinha, J. Luo, S. Rajagopalan, S. Batterywala, N. V. Shenoy, and H. Zhou, "Impact of Modern Process Technologies on the Electrical Parameters of Interconnects," in *Proc. VLSI Des.,* Jan. 2007, pp. 875-880.

[84] Ankur Jain, "Three-dimensional (3D) Technology: An Overview of Challenges and Opportunities" *3D IC Design and Architecture Workshop,* in Hsinchu, Taiwan, Sep. 2008.