

國立臺灣大學電機資訊學院資訊工程學研究所

博士論文

Graduate Institute of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Doctoral Dissertation

多核心系統晶片之系統合成方法

System Synthesis for Multi-Processor System-on-Chips

陳依蓉

Yi-Jung Chen

指導教授：楊佳玲 博士

Advisor: Yang Chia-Lin, Ph.D.

中華民國 99 年 7 月

July, 2010

多核心系統晶片之系統合成方法

研究生：陳依蓉

指導教授：楊佳玲 博士

國立台灣大學資訊工程研究所

中文摘要

近年來由於製成技術的進步，使支援多個工作同時進行之多核心架構 (Multi-core architecture) 成為晶片設計主流。多核心架構因系統上有多個可獨立運做之運算單元 (Processing Elements, PE)，使其特別適用於有大量平行度之應用程式。但每個可獨立運做之 PE 也會同時發出記憶體存取之需求，進而對記憶體系統造成不小之壓力。因此在多核心架構晶片設計上，記憶體系統之設計對整體系統效能有相當重要的影響。因此，在本論文中，我們針對兩種不同的多核心系統晶片架構：(1) 以傳統二維方式連結處理器及動態存取記憶體 (Dynamic Random Access Memory, DRAM) 多核心系統晶片 (Multi-Processor System-on-Chips (MPSoCs) with traditional 2D CPU-DRAM connection)，及(2) 以三維堆疊 DRAM 之多核心系統晶片 (MPSoCs with stacked DRAMs)，分別提出考量記憶體系統架構之系統合成方法。

在 MPSoCs with traditional 2D CPU-DRAM connection 架構方面，我們發現，為達最佳系統效能，系統晶片上之運算單元與記憶體模組之資源分配應針對所執行之應用程式，以避免記憶體系統成為系統效能之主要瓶頸。然而，傳統之多核心單晶片系統設計流程中，運算單元與記憶體模組之資源分配通常都分開獨立進行，因此無法考量到兩種資源之分配多寡對系統之影響。因此，在本論文中，我們針對此一問題提出第一個運算單元與記憶體系統資源共同合成之多核心單晶片系統設計流程 (PE and Memory Co-Synthesis (PM-COSYN) for MPSoCs)。

在以三維堆疊技術實現之單晶片多核心系統方面，由於三維堆疊技術可利用穿矽通孔 (Through-Silicon Vias, TSVs) 所組成之垂直通道 (Vertical Bus)，使 DRAM 可與運算處理器晶片以三維堆疊的方式整合在同一晶片系統上，此外，TSV 可高密度地擺放在晶片上進而提供大量記憶體頻寬 (Memory Bandwidth)。因此有大量 Memory Bandwidth 需求的 PE，可整合記憶體控制器 (DRAM Memory Controller,

DMC)於其上，使 PE 可透過近端整合之 DMC 與堆疊其上之 DRAM 溝通。由此可見，MPSoCs with stacked DRAM 中 PE 與 DRAM 之溝通介面為由多個 DMC 所組成之分散式介面(Distributed DRAM Interface)。然而，DMC 所需之晶片資源相當多，如果 PE 可不近端整合 DMC，其資源可以用來擴大 PE 之靜態存取記憶體(Static Random Access Memory, SRAM)之容量。此外，TSV 雖可提供大量 Memory Bandwidth，但 TSV 的製成除需額外之金錢花費外，更對晶片良率(Chip Yields)有負面的影響。因此，我們在本論文中，我們針對 MPSoCs with stacked DRAMs，提出一套分散式記憶體系統介面合成方法(Distributed Memory Interface Synthesis)。此合成方法根據系統對記憶體系統進行存取的行為與需求，決定晶片上之記憶體控制器個數，以及每個記憶體控制器之 Vertical Bus 寬度，讓系統可維持在指定之效能需求下，使晶片上之 TSV 總數量最少化。

關鍵字 – 運算單元，記憶體系統，多核心單晶片系統，合成，三維堆疊，分散式記憶體系統介面



SYSTEM SYNTHESIS FOR MULTI-PROCESSOR SYSTEM-ON-CHIP

Student: Yi-Jung Chen Advisors: Dr. Chia-Lin Yang

Department of Computer Science and Information Engineering
National Taiwan University



Abstract

Multi-core architecture is attractive to applications with significant parallelism since multiple processing elements (PEs) are put on a single die to support parallel execution. However, multi-core architecture also stresses the memory system with concurrent memory accesses from different PEs. With the number of cores on a chip increases, the main memory bandwidth requirement also grows. Therefore, it is important to have a memory-aware design when designing Multi-Processor System-on-Chips (MPSoCs). In this thesis, we propose memory-aware MPSoC synthesis methods for MPSoCs with two different architectures: (a) MPSoCs with the traditional 2-Dimensional (2D) CPU-DRAM connection, and (b) MPSoCs with 3-Dimensional (3D) stacked DRAMs. For MPSoCs with the traditional 2D CPU-DRAM connection, the main memory bandwidth is limited due to pin limitations. To maximize system performance, it is important to simultaneously consider the PE and on-chip memory architecture design with limited on-chip resource. That is, on one hand, we want to allocate as many PEs as possible to fully utilize the available task parallelism in the target applications, and on the other hand, we need to incorporate a significant amount of on-chip memory to alleviate memory bottleneck. However, in a traditional MPSoC design flow, memory and computation components

are often considered independently. To tackle this problem, we develop the first PE and memory co-synthesis framework for MPSoCs with 2D CPU-DRAM connections. The goal of the algorithm is to simultaneously synthesize the allocation of PE and on-chip memory modules so that system performance is maximized subject to the resource constraint. In MPSoCs with stacked DRAMs, the 3D die-stacking technology utilizes Through-Silicon Vias (TSVs) to integrate processing cores and DRAMs on the same chip. Moreover, the TSVs that can be placed densely provide high DRAM bandwidth for the system. Therefore, to utilize the high DRAM bandwidth, each PE can have a local DRAM memory controller (DMC) so that it can directly access the DRAM module stacked on top of the PE. This forms a distributed memory interface for CPU-DRAM connection in MPSoCs with stacked DRAMs. However, a DMC occupies a significant share of transistor budget, which can be traded for enlarging the capacity of high speed local SRAM. Moreover, TSVs need extra manufacturing cost and have adverse impact on chip yields. Therefore, the distributed memory interface, including the number of allocated DMCs and vertical bus width of each DMC, should be designed carefully. To tackle this problem, in this thesis, we propose the first algorithm to synthesize the DMC allocation and vertical bus allocation for MPSoCs with stacked DRAMs. The goal of the proposed algorithm is to find a proper distributed memory interface design for the given task set so that the total number of TSVs in the system is minimized while the user-defined performance constraint is met.

Keywords— Processing Elements, Memory Subsystem, Multi-Processor System-on-Chip, Synthesis, 3-Dimensional Integration, Distributed Memory Interface

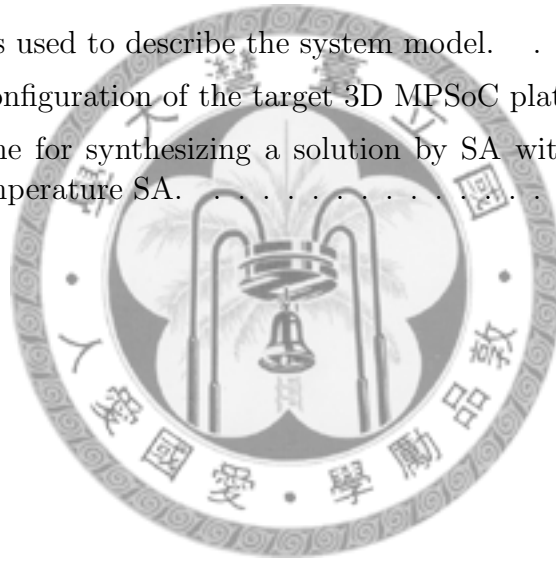
Table of Contents

Abstract	i
List of Tables	v
List of Figures	vi
Chapter 1. Introduction	1
1.1 PE and Memory Co-Synthesis for MPSoCs with Traditional 2D CPU- DRAM connection	3
1.2 Memory System Synthesis for MPSoCs with 3D Integration	5
1.3 Organization of the Thesis	6
Chapter 2. Related Works	7
2.1 PE and Memory System Design for Traditional 2D MPSoCs	7
2.2 System Design for 3D ICs	9
2.2.1 Architecture Design with 3D Technologies	9
2.2.2 Cost modeling of 3D technologies	11
2.2.3 Thermal issues of 3D ICs	12
Chapter 3. PE and Memory Co-Synthesis for 2D MPSoCs	14
3.1 Introduction	14
3.2 System Model and Problem Formulation	16
3.2.1 System Model	16
3.2.2 Problem Formulation	18
3.3 PM-COSYN for MPSoCs with 2D CPU-DRAM Connection	18
3.3.1 PE&Memory Allocation	20
3.3.2 Memory Reduction Process	24

3.4	Experimental Results	24
3.4.1	Experimental Setup	24
3.4.2	Analysis of PM-COSYN	26
3.4.3	Comparison with Simulated-Annealing Optimizer	31
3.4.4	Scalability Issue	32
Chapter 4. Distributed Memory Interface Synthesis for MPSoCs with Stacked DRAM		33
4.1	Introduction and Motivation	33
4.2	Overview of 3D Integration	38
4.3	System Specifications and Problem Formulation	39
4.3.1	System Model	39
4.3.2	Problem Formulation	40
4.4	Memory System Synthesis Algorithm for MPSoCs with Stacked DRAM	43
4.5	Experimental Results	46
4.5.1	Experimental Setup	46
4.5.2	Analysis of Experimental Results	49
Chapter 5. Concluding Remarks and Future Work		54
5.1	PE and Memory Co-Synthesis for Traditional 2D NoCs	54
5.2	Memory System Design for MPSoCs with Stacked DRAM	55
Bibliography		56
Publication List		64

List of Tables

3.1	Task set properties.	25
3.2	Detail configuration of target NoC platform.	26
3.3	Comparison of PM-COSYN and PM-SA.	30
3.4	Comparison of PM-COSYN and PM-SA.	31
4.1	Variables used to describe the system model.	41
4.2	Detail configuration of the target 3D MPSoC platform.	48
4.3	CPU time for synthesizing a solution by SA with and without the Low-Temperature SA.	53



List of Figures

1.1	Intel processor evolving with Moore's Law [1].	2
1.2	Memory traffic as the number of chip-multiprocessor cores varies in the next technology generation. [55].	3
1.3	Key technologies of 3D stacking. [60]	4
1.4	Baseline architecture of 3D MPSoC with stacked DRAM.	6
3.1	Architectural overview of a 2D NoC.	15
3.2	PM-COSYN overview.	19
3.3	Pseudo code of Victim PE Selection.	21
3.4	Pseudo code of Data Block Assignment.	23
3.5	Number of on-chip memory allocated for highly parallel task sets with various trans.bit/task_cycle.	27
3.6	Normalized system execution time of systems with various number of on-chip memories: (a) task set t0, and (b) task set t3.	28
3.7	Normalized execution time of systems with various number of on-chip memories: (a) Consumer+Telecom, and (b) Mpeg2.Enc+Mpeg2.Dec.	30
3.8	How PM-COSYN program execution time scales with task set set size.	32
4.1	Baseline architecture of MPSoCs with stacked DRAMs: (a) Architectural overview of MPSoCs with stacked DRAM, and (b) Detailed architecture of a DRAM controller.	35
4.2	5 different DRAM partitions for different numbers of DMC allocation in a 16-tile MPSoC with stacked DRAM.	36
4.3	Illustration of trading the transistor budget of a DMC to enlarge the local SPM.	37
4.4	TSV connection yield calculated by equation of poisson distribution [47].	37
4.5	Flow of the SA-based memory system synthesis algorithm.	44
4.6	Flow of the Low-Temperature SA algorithm.	47

4.7	Synthesis results for task sets with different average data transfer per edge and performance constraint is set to no performance degradation compared the baseline: (a) Number of allocated DMCs and TSVs, and (b) TSV reduction compared to the baseline architecture.	50
4.8	Synthesis results for task sets with different average data transfer per edge and performance constraints are set to 0% to 9% performance degradation compared the baseline architecture: (a) Number of allocated DMCs and TSVs, and (b) TSV reduction compared to the baseline architecture.	51
4.9	Distributed memory interface synthesis results of Consumer+Telecom with performance constraint set to 3% degradation compared to the baseline.	52
4.10	Comparison of SA engine with and without Low-Temperature SA. . .	53



Chapter 1

Introduction

The relentless pace of technology leads to multi-core microprocessor designs with extensive on-die integration of large number of cores. As predicted by the Moore's Law, the number of transistors that can be inexpensively placed on an integrated circuit is doubling approximately every two years [49]. Figure 1.1 shows the number of transistors integrated into Intel processors. We can see that multi-core architectures have become mainstreams in the last few years. As the technology continues to shrink, the number of cores on a chip will continue to grow. With this trend, Multi-Processor System-on-Chips (MPSoCs) are becoming a popular solution to meet the growing processing demands of embedded applications.

Multi-core architecture improves system performance by putting multiple Processing Elements (PEs) on the system to exploit task parallelism. However, concurrent memory accesses from different PEs also stress the memory subsystem. Due to the performance scaling discrepancy between DRAMs and processing cores, the DRAM system is becoming the performance bottleneck with technology scaling, and this is known as the Memory Wall issue [45,68]. With multiple PEs competing for the DRAM bandwidth, the memory wall issue is further worsened. In [55], authors provide a thorough study on how the memory wall problem restricts future multi-core architecture scaling. Figure 1.2 is one of the results of [55], and it shows how the off-chip memory traffic varies as the number of chip-multiprocessor cores increases. With the same chip area and considering the next technology generation, we can see that, the off-chip memory traffic grows super-linearly as the number of

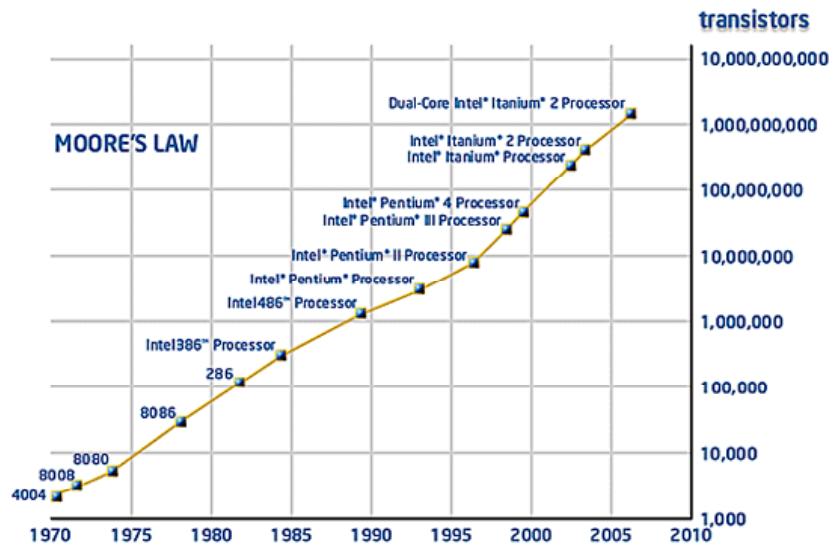


Figure 1.1: Intel processor evolving with Moore's Law [1].

processor cores increases. Therefore, from the above discussion, we can see that one critical issue in an MPSoC design is how to deal with the memory wall issue.

Currently, the most commonly seen architecture is MPSoCs with the traditional 2D CPU-DRAM connection. In this kind of architecture, the off-chip DRAM bandwidth is limited by pin count of the chip. Therefore, one of the major design issue for MPSoCs with the traditional 2D CPU-DRAM connection is to how to minimize the number of off-chip memory accesses to alleviate the DRAM bottleneck problem.

Recently, the emerging 3D die-stacking technology is proposed as one of solution for solving the DRAM bandwidth issue [30,67]. 3D die-stacking uses Through-Silicon Vias (TSVs) as the vertical interconnection to stack several active devices on the same chip in the third dimension. 3D die-stacking has the properties of heterogeneous integration and the TSVs can be densely placed on a chip. Therefore, with 3D die-stacking, DRAMs can be stacked directly on top of PEs and large DRAM bandwidth can be supported by the high-density vertical links. In MPSoCs with stacked DRAMs, PEs with large DRAM bandwidth requirement can have their own local DRAM memory controllers (DMCs) so that they can access the DRAM

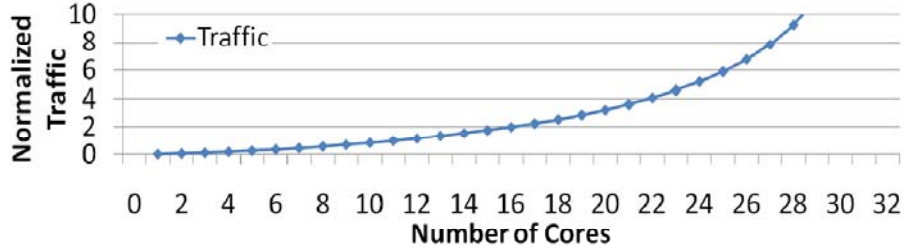


Figure 1.2: Memory traffic as the number of chip-multiprocessor cores varies in the next technology generation. [55].

stacked on top it directly through the local DRAM interface [42]. As we can see, MPSoCs with stacked DRAMs has a distributed memory interface that allows more than one DMC on the system, and one of the major design issue in such architecture would be how to design the distributed memory interface, such as the number of DMCs and the vertical bus width of each DMC, according to the behavior of the target application set.

Therefore, in this thesis, we develop two memory-aware system synthesis algorithms for MPSoCs with the traditional 2D CPU-DRAM connection and MPSoCs with stacked DRAMs, respectively. The two algorithms are (1) *PE and memory co-synthesis (PM-COSYN) algorithm or MPSoCs with 2D CPU-DRAM connection*, and (2) *Distributed memory interface synthesis for MPSoCs with stacked DRAMs*. More details are discussed as follows.

1.1 PE and Memory Co-Synthesis for MPSoCs with Traditional 2D CPU-DRAM connection

As mentioned earlier, for MPSoCs with traditional 2D CPU-DRAM connection, the off-chip memory bandwidth is limited by the pin count of the chip. One way to mitigate the off-chip memory bandwidth problem would be minimizing the number of off-chip memory accesses. To achieve this, we can incorporate a signifi-

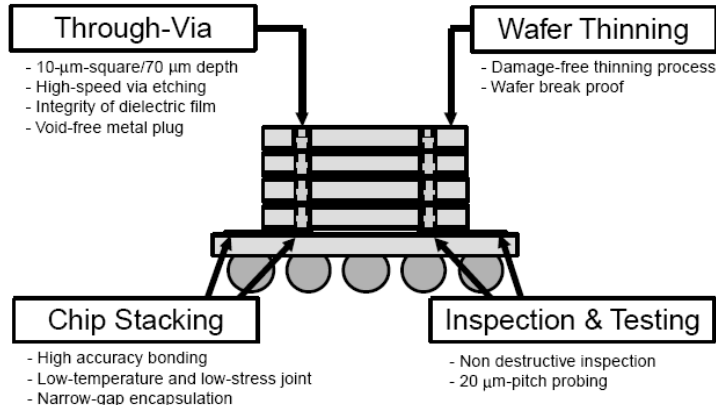


Figure 1.3: Key technologies of 3D stacking. [60]

cant amount of on-chip memory modules [55]. However, we also want to dedicate as many die resources to PEs as possible to fully utilize the available task parallelism in the target applications. As we can see, one critical design issue in MPSoCs with 2D CPU-DRAM connection is how to utilize the limited die resource to achieve a balanced design between memory and computation subsystems.

To tackle this problem, in this thesis, we propose the first PE and memory co-synthesis (PM-COSYN) framework. The target architecture discussed in PM-COSYN is a tile-based Network-on-Chip (NoC) with the traditional 2D CPU-DRAM connection. Each tile in the NoC can contain either a PE or an on-chip memory module. The goal of PM-COSYN is to simultaneously allocate PEs and on-chip memory modules for MPSoCs so that system performance is maximized and the area constraint is met. Since the allocation problem are known to be NP-complete for distributed systems [21], we propose a greedy-based heuristic to solve the PE and memory allocation for 2D MPSoCs. Starting with an initial setting where the number of PEs is equal to the degree of task parallelism, and memory modules are allocated to the remaining tiles, PM-COSYN adopts a greedy-based iterative method to refine the system configuration. During the refinement process, PEs are gradually replaced with on-chip memory modules to see if it results in better system performance. Details of the PM-COSYN framework are described in Chapter 3.

1.2 Memory System Synthesis for MPSoCs with 3D Integration

To mitigate interconnect-related problems in deep submicron, the emerging 3D integration technology has been proposed. Figure 1.3 shows a chip with 3D integration technology. Several active devices are connected to each other by the vertical interconnects implemented by Through-Silicon Vias (TSVs) or Through-Vias. 3D die-stacking has the properties of (1) heterogeneous integration that allows chips with different technologies, such as DRAM and processing cores, to be integrated at different layers of the same chip, (2) short vertical link that shortens the average interconnect length and improves system performance [37], and (3) high density TSVs that support large DRAM bandwidth, e.g. over three hundreds 1K-bit CPU-DRAM buses on a 1cm^2 chip [39]. Due to these properties, 3D die-stacking enables a memory processor interconnect that is both very high bandwidth and low latency [30]. Recently, inspired by 3D die-stacking, the architecture of MPSoCs with stacked DRAMs and distributed memory interface as shown in Figure 1.4 has been discussed [39, 42, 44]. Processing cores are placed on the logic layer, and cores are connected through Network-on-Chip with 2D mesh topology. Each tile is composed of a PE, a DMC, a local scratch-pad memory (SPM) module, and a router for connecting the NoC. One or more DRAM memory layers are stacked on top of the logic layer. Each PE can directly access the DRAM module stacked on-top of it through its local DMC and the vertical bus implemented by TSVs. PEs can address DRAM modules on top of other PEs by transporting the request and data through the horizontal NoC.

However, a DMC also takes a lot of transistor budget since it utilizes several queues to store the read/write data. When a PE does not have large bandwidth requirement, the transistor budget of DMC can be used to enlarge the capacity of local SPM so that more data can be stored in the short latency SPM. But at the same time, each DMC can only have a limited data bus width due to the area constraint so more DMCs are needed when the system needs higher DRAM bandwidth. Moreover, although more TSVs provides higher DRAM bandwidth, more manufacturing cost is also needed [66] and chip yields may be degraded [47]. Therefore, a major issue in

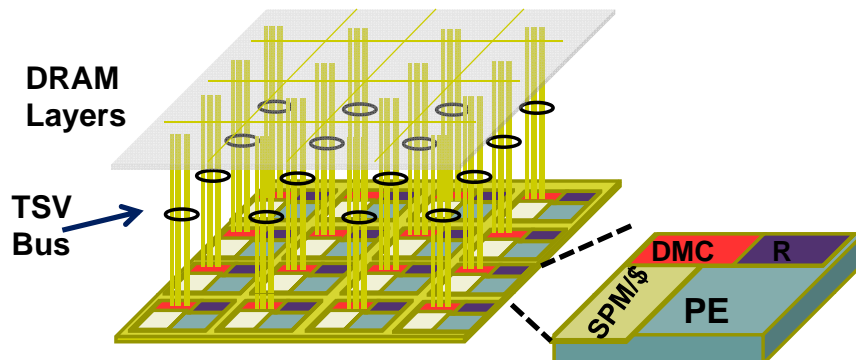


Figure 1.4: Baseline architecture of 3D MPSoC with stacked DRAM.

MPSoCs with stacked DRAM is how to find a distributed memory interface design, e.g. the number of allocated DMCs and the vertical bus width of each DMC, that meets requirements of the target application set. To tackle this problem, in this thesis, we propose the first distributed memory interface synthesis algorithm for MPSoCs with stacked DRAM modules. The goal of the proposed algorithm is to find a distributed memory interface design that minimizes the TSV cost while the user-defined performance constraint is met. Due to the large solution space, we proposed a Simulated-Annealing (SA) [34] based synthesis algorithm. Details of the proposed 3D memory system synthesis algorithm 3D MPSoCs are described in Chapter 4.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. The discussion of the related work is presented in Chapter 2. Chapter 3 shows the PE and memory co-synthesis algorithm for MPSoCs with traditional 2D CPU-DRAM connection. Chapter 4 presents the distributed memory interface synthesis algorithm for MPSoCs with stacked DRAMs. Conclusions and future works are given in Chapter 5.

Chapter 2

Related Works

In this chapter, we present previous works that are related to the proposed PM-COSYN framework. The previous works are of two directions: (a) PE and memory system design in traditional 2D MPSoCs, and (b) System design for 3D ICs.

2.1 PE and Memory System Design for Traditional 2D MP-SoCs

Several works have been proposed to perform system-level designs for application-specific MPSoCs [11] [16] [31]. In [16], Dick et al. proposed a multi-objective genetic algorithm for synthesizing hardware and software architecture of an embedded system. Hong et al. [11] proposed an SA-based synthesis algorithm that decides PE allocation, task assignment, tile mapping and task scheduling of an NoC with regular mesh. However, in [11] and [16], only PE allocation is considered. Kim et al. [31] described real chip implementation issues of NoCs and proposed a memory-centric NoC chip design for homogeneous MPSoCs.

Several previous works are proposed to simultaneously optimize memory and communication system of MPSoCs. In [52], a algorithm for memory and communication resource allocation is proposed for bus-based MPSoCs. Similar to [52], Kim et al. [33] propose a simulation-based methodology to explore the on-chip bus architectures and memory allocation for MPSoC system. The simulation efficiency of each configuration is optimized by using a static performance estimation to prune

the design space drastically in the first step, and then using a trace-driven simulation to evaluate each configuration. Issenin et al. [29] proposed to allocate memory resource and synthesize communication architecture in a mesh-based NOC by a data reuse analysis approach. In [52] and [29], PE allocation is predefined, and only memory and communication resource is synthesized.

Another kind of work is to simultaneously synthesize the software architecture and data allocation for the given MPSoC platform. Target at homogeneous chip-multiprocessor, where each tile has a shared of scratch-pad memory (SPM), Chen et al. [10] proposed an compiler approach to decide task allocation and data allocation for the applications running on the platform. Given an application model written in SystemC TLM 2.0, Lukasiewyca et al. [43] proposed an approach to fully automatic synthesize resource allocation, task binding, data mapping, and transaction routing for MPSoC platforms.

Several works are proposed to manage the memory system of MPSoCs. Since software controlled scratchpad memories (SPMs) have highly predictable memory access behavior, they have been proposed as an alternative to large L2 caches [5], especially for embedded systems that usually have the real-time requirement. Therefore, several works have been proposed to decide which contents should be assigned to the on-chip SPM. Targeting at SoCs with both on-chip SPM and cache memory, Panda et al. [51] proposed a method to decide which data should be mapped to the SPM, and which data should be mapped to the off-chip memory, so that system performance is optimized. The proposed method carefully partitions the scalar and array variables, and takes the lifetimes of different variables into consideration. Udayakumaran et al. [63] propose a dynamic allocation methodology for global and stack data, and program code to scratch-pad memory. The proposed method accounts for changing program requirements at runtime, and yields 100% predictable memory access times, which is very critical to hard real-time systems. In [48], Monchiero et al. proposed an on-chip hardware memory management unit to perform data allocation/deallocation on the physically distributed and logically shared on-chip memory space of an MPSoC.

Our PM-COSYN differs from these prior works in that it focuses on simul-

taneously allocating PE and on-chip memory for NoC with the area constraint, and the previous works consider the allocation of the two kinds of resource independently or focus on further optimization of the given platform.

2.2 System Design for 3D ICs

Due to the development of the Through-Silicon Via (TSV) technology, the 3D stacking technology has been one of the most promising method to continue the scaling of Moore's law. In recently years, abundant research works are proposed to discuss different aspects of 3D technologies. Therefore, in this section, we categorize previous works related to 3D IC research into (1) Architecture design with 3D technologies, (2) Cost modeling of 3D technologies, and (3) Thermal issues of 3D ICs.

2.2.1 Architecture Design with 3D Technologies

In the architecture design of 3D ICs, we can divide into two broad categories, pure chip stacking and true 3D design. In the first category, different chips are purely stacked vertically to reduce inter-module communication latency by short latency TSVs, and most of the recent works falls in this category. Kgil et al. [30] proposed PicoServer, a CMP architecture that employs 3D technology to bond one die containing several simple slow processing cores to multiple DRAM dies sufficient for a primary memory. Since TSVs enables the wide and low-latency DRAM buses and provide sufficient memory bandwidth, in [30], they propose to remove the large shared L2 cache, and replace by more additional simple processing cores to allow more threads executing in parallel. In [38], Lui et al. showed that stacking DRAM on processor cores achieves lower memory access latency and system energy consumption, when comparing to traditional 2D systems that has DRAM as an off-chip memory. They also showed that, the performance and energy advantage of 3D stacking comes from the short, vertical, and non-pin limited on-chip connections implemented by TSVs. Target at the DRAM on processor architecture, several works are further proposed to discuss how to design the memory system. Since

stacking DRAM on processor allows more frequent processor to DRAM accesses, and chip temperature may increase due to die-stacking, Ghosh et al. [22] revisits the refresh interval of stacked DRAM. They propose a novel refresh technique to eliminate all the unnecessary DRAM refresh overheads in a 3D IC. In [67], proposed to re-architect the memory hierarchy, including the L2 cache and DRAM interface, so that it can take full advantage of the massive bandwidth of 3D die-stacking. They proposed a technique called SMART-3D, which is a new 3D-stacked memory architecture with a vertical L2 fetch/write-back network using a large array of TSVs. The basic idea of SMART-3D is to leverage the TSV bandwidth to hide latency behind very large data transfers. Loi et al. [42] target at many-core platform with stacked DRAM and propose a distributed memory controller architecture. Moreover, they proposed a memory controller design for exploiting the highly efficient vertical bus implemented by TSVs. Marongiu et al. [44] proposed a programming framework with compiler support that help application designers to fully exploit the potential of vertically stacked memory. The proposed framework targets at an explicitly managed 3D-stacked memory hierarchy, which requires placement of data across multiple vertical memory stacks to be carefully optimized. Different from the above works that stack DRAM with processor cores, Sun et al. [59] proposed to integrate Magnetic Random Access Memory (MRAM) atop conventional CMPs. In [18], Dong et al. further discussed the circuit design issues for MRAM, and proposed a MRAM cache model. Based on the model, they compared MRAM against SRAM and DRAM in terms of area, performance and energy. In [4], Awasthi et al. consider 3D organizations of a single-threaded clustered micro-architecture to understand how floorplanning impacts performance and temperature. They looked into three different stacking methods, cache-on-cluster, cluster-on-cluster and staggered. They found that, the delays between cache and ALUs are the most critical to performance, and showed that a word-interleaved cache with a staggered 3D placement achieves the best balance between temperature and performance. In [7], Black et al. studied the performance advantages and thermal challenges of two forms of die stacking: stacking a large DRAM or SRAM cache on a microprocessor and dividing a traditional micro-architecture between two die in a stack. Another issue in stacking processor cores with different memory modules is the design of interconnec-

tion network. Li et al. [37] proposed a router architecture and topology design that makes use of a network architecture embedded into the L2 cache memory. They also showed that, a 3D L2 memory architecture generates much better results than the conventional 2D designs under different number of layers and vertical connections. In [56] and [50], a design tool for synthesizing application-specific interconnection network for 3D NoCs is proposed. The tool determines the best NoC topology for the target application, finds paths for the communication flows, assigns routers to the 3D layers, and place them in each layer. The optimization goal is to optimize performance subject to the given TSV link constraints.

Another category of 3D chip design is true 3D design. In true 3D designs, major components of a system are re-designed to allow logic gates or modules of an individual components to be interconnected through TSVs. The idea of this design is to fully explore the low latency of TSVs. Vaidyanathan et al. [64] implemented a few components of a microprocessor, such as adder and tag drive, using custom design to show the potential performance and power benefits achievable through 3D integration under thermal constraints. They also introduced a standard cell based 3D design flow which leverages the commercial 2D design tools. Loh [39] explored more aggressive 3D DRAM organizations that make better use of the additional die-to-die bandwidth provided by 3D stacking. They also identified that the significant increase in memory system performance makes the L2 miss handling architecture (MHA) a new bottleneck. Tsai et al. [62] explored the architectural design of cache memories using 3D circuits. Moreover, they proposed a delay and energy model called 3DCacti to explore different 3D design options of partitioning a cache. Black et al. [8] demonstrated that a complex device as an iA32 microprocessor can be repartitioned or split between two die in order to simultaneously improve performance and power. The 3D structure of iA32 is examined and applied to a real x86 deeply pipelined high performance microprocessor.

2.2.2 Cost modeling of 3D technologies

Since 3D is an emerging technology, the tradeoffs of using the technology for implementation should be fully studied so that system designers can find a best de-

sign. In [57], a modeling technique for a priori cost and performance estimations for mixed-signal system implementations is proposed. In [46], a yield and cost modeling for 3D chip stack technologies is proposed. However, in this modeling, the impact of TSVs is not discussed. Weerasekera et al. [66] discussed realistic metrics for performance and cost trade-offs both at implementation phase and conceptual level for verification in the 3D integration technology. Kim et al. [32] studied the impact of TSV on various aspects of 3D layouts, including the silicon area of TSV, wire length and TSV count. Dong et al. [19] proposed a system-level cost analysis tool to help designers to determine if the 3D integration method is a cost effective technology for a particular IC design. In [65], a complete set of self-consistent equations including self and coupling terms for resistance, capacitance and inductance of various TSV structures is proposed.

2.2.3 Thermal issues of 3D ICs

Since stacking multiple chips on the third dimension is bad for heat flow and cause serious thermal issues, several works are proposed to deal with the thermal problem of 3D ICs. One kind of method is to use proper placement [24, 25] or floorplanning [12] of chip modules to minimize the maximum temperature of the chip. Goplen et al. [24] proposed a thermal placement method that uses an iterative force-directed approach in which thermal forces direct cells away from areas of high temperature. In [25], the same authors proposed analytical and partitioning-based techniques to explore the tradeoff between wire-length, inter-layer via counts, and thermal effects. Cong et al. [12] proposed a new 3D floorplan representation so that the solution space can be efficiently explored, and an efficient thermal-driven 3D floorplanning algorithm with an integrated compact resistive network thermal model. Another kind of method is to use task scheduling techniques to avoid modules on adjacent area are not active at the same time [58, 70]. Sun et al. [58] proposed a 3D MPSoC thermal optimization algorithm that conducts task assignment, scheduling, and voltage scaling. Detailed thermal analysis is used to guide a hotspot mitigation algorithm that incrementally reduces the peak MPSoC temperature by appropriately adjusting task execution times and voltage levels. Zhou et al. [70] proposed an OS-

level scheduling algorithm that performs thermal-aware task scheduling on a 3D chip. The proposed algorithm leverages the inherent thermal variations within and across different tasks, and schedules them to keep the chip temperature low.



Chapter 3

PE and Memory Co-Synthesis for 2D MPSoCs

In this chapter, we present the proposed PE and Memory Co-Synthesis (PM-COSYN) algorithm for MPSoCs with the traditional 2D CPU-DRAM connection. We first present the motivation and the introduction on the baseline architecture discussed here. Next, we present the proposed PM-COSYN algorithm, and some preliminary experimental results.

3.1 Introduction

In this thesis, we propose the first PE and Memory Co-Synthesis (PM-COSYN) framework for MPSoCs with the traditional 2D CPU-DRAM connection. For such an MPSoC with limited die area, one critical design issue is how to utilize the available resources to achieve a balanced design between memory and computation systems. That is, on one hand, we want to dedicate as many die resources to PEs as possible to fully utilize the available task parallelism in the target applications, and on the other hand, we need to incorporate a significant amount of on-chip memory to alleviate memory bottleneck. Therefore, the goal of PM-COSYN is to simultaneously synthesize PE and on-chip memory allocation to optimize system performance for MPSoCs with the area constraint.

The target architecture discussed in PM-COSYN is a tile-based Network-on-Chip (NoC) as shown in Figure 3.1 [35]. A tile can contain either a PE or a memory module. Each resource connects to its local switch through its routing

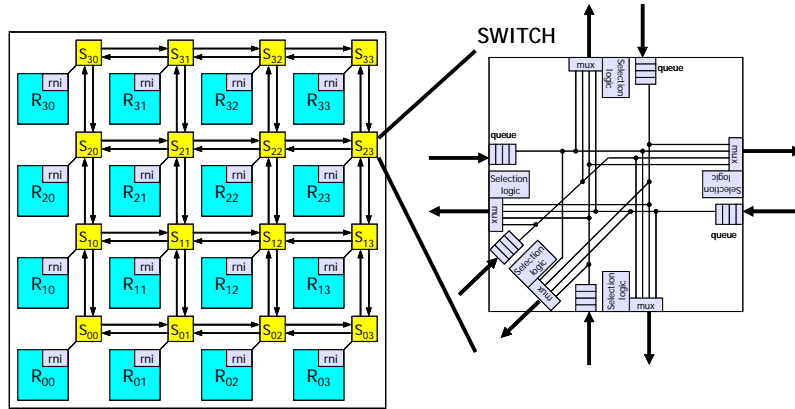


Figure 3.1: Architectural overview of a 2D NoC.

network interface (rni). A switch routes and buffers messages between resources. As shown in Figure 3.1, each switch is connected to one resource and four neighboring switches, and each resource is connected to one switch. With this architecture, the communication among tiles is achieved by sending packets to one another over the network instead of routing wires.

Target at the NoC architecture shown in Figure 3.1, PM-COSYN simultaneously decides the PE and on-chip memory allocation for the given NoC template such that the execution time of the target task set is minimized. Starting with an initial setting where the number of PEs is equal to the degree of task parallelism, and memory modules are allocated to the remaining tiles, PM-COSYN adopts a greedy-based iterative method to refine the system configuration. During the refinement process, PEs are gradually replaced with memory modules to see if it results in better system performance.

We evaluate PM-COSYN on a set of synthetic task sets with the same control and data flow graph but different data access ratios. The experimental results show that with the same level of task parallelism, different memory access ratio does result in different optimal system configuration. For the tested task sets, the required memory modules vary from 1 to 7 in a 3×3 NoC. The experimental results show that PM-COSYN successfully identifies the correct numbers of PE and

memory module for optimal system performance. We also test PM-COSYM on a set of real-world workloads. We construct a workload mix including consumer and telecommunication applications from EEMBC [20] that are typical to a mobile device. We find that this set of workloads presents both significant amount of task-level parallelism and memory accesses. For a 3×3 NoC, the optimal system configuration is 7 PEs and 2 memory modules. This configuration improves system performance by 13.94% compared to 9-PE configuration. We also compare PM-COSYN to a Simulated-Annealing (SA) method. Experimental results show that PM-COSYN generates a better solution in shorter CPU time compared to the SA method. This demonstrates the proposed PM-COSYN framework is effective in solving the PE-memory co-synthesis problem for a NoC-based MPSoC. The details of PM-COSYN are presented in the following subsections.

3.2 System Model and Problem Formulation

In this section, we present the models used to describe the target application set and NoC architecture. We also present the formal problem definition of PM-COSYN.

3.2.1 System Model

Our system consists of an application model and an NoC architecture model. Details of the two models are described as follows.

Application Model

We represent the target application set by a Data Flow Graph (DFG). The DFG considered in this thesis is a directed acyclic graph in which a node is a task, and the directed edges represent transfers of *data blocks*. A data block is the collection of scalars or arrays which is similar to the definition used in [52]. We use $G = \langle V, E \rangle$ to denote a DFG, where V represents the set of tasks and E represents the set of directed edges. Each vertex $v_i \in V$ has the following properties:

- $c(v_i)$ denotes the number of cycles that v_i used to complete on the reference PE.
- $p(v_i)$ denotes the priority of v_i . We adopt List Scheduling [3] as our baseline scheduler to schedule tasks allocated on the same PE. In List Scheduling, tasks are scheduled according to their precedence relations and priorities. We assume the priorities are given by system designers in advance.

Each $e_i \in E$ represents a data block transfer. Each e_i is associated with $d(e_i)$, which denotes the ID of the data block that is transferred by e_i . For each application set, there is a data block library $D = \{d_1, \dots, d_k\}$ to store all the data blocks touched in the application set, and d_i indicates the i -th data block in the library. Each d_i is associated with $size(d_i)$, which indicates the size of data block d_i (in bits).

NoC Model

The NoC architecture under consideration is composed of $m \times n$ regular tiles interconnected by a 2D mesh network. We model such an NoC-based system with $m \times n$ tiles as an *Architecture Graph* $N = \langle T, L \rangle$, which is a *directed graph*, where $T = \{t_1, \dots, t_{m \times n}\}$ is the set of tiles and L is the set of links between tiles. Each link $l_{i,j} \in L$ represents a link connection between t_i and t_j and is associated with $w(l_{i,j})$ which stands for the link width of $l_{i,j}$. Each tile can contain either a PE or a memory module. In our algorithm, we assume that all PEs allocated on the NoC are general-purpose processor with the same micro-architecture, and each PE has a local buffer to store the most recently accessed data. Therefore, allocating tasks with shared data on the same PE would help the reduction of on-chip communication traffic. Each on-chip memory module has the capacity that a tile can accommodate with.

Similar to [28], we also assume a *static XY* routing scheme [23] as our underlying routing protocol. The static XY routing scheme first routes packets along the X -axis. Once it reaches the column where the destination tile lies in, the packet is then routed along the Y -axis. Note that the proposed algorithm can be applied on NoCs with any kinds of routing algorithm.

3.2.2 Problem Formulation

The goal of PM-COSYN is to simultaneously synthesize PE and on-chip memory allocation for application-specific NoCs with the area constraint. Since NoCs are usually composed of regular tiles, in this thesis, the area constraint is specified by the tile number of the target NoC platform. As we can see, the assignment of tasks and data blocks should be adjusted when the allocation of PE and on-chip memory changes. Therefore, for a given DFG $G = \langle V, E \rangle$, a data block library $D = \{d_1, d_2, \dots, d_k\}$, and an NoC architecture template $N = \langle T, L \rangle$, we can define the problem as follows.

Given $G = \langle V, E \rangle$, $D = \{d_1, \dots, d_k\}$ and $N = \langle T, L \rangle$
Find P , M and D' , and the function ϕ and ω ,
 such that T_{sys} is minimized
Subject to $|P| + |M| = |T|$

In the problem formulation, P denotes the set of allocated PEs and M denotes the set of on-chip memories. D' denotes the set of data blocks assigned to on-chip memory modules, where $D' \subseteq D$. T_{sys} denotes the system execution time. After the allocation of PE and on-chip memory is decided, we have to consider how to assign tasks and data blocks to these selected modules. We use the function $\phi : V \rightarrow P$ and $\omega : D' \rightarrow M$ to represent the assignment of tasks and data blocks, respectively.

3.3 PM-COSYN for MPSoCs with 2D CPU-DRAM Connection

In this section, we describe the details of the proposed PE-Memory Co-Synthesis (PM-COSYN) algorithm. PM-COSYN is a greedy-based iterative algorithm, and the overview of PM-COSYN is shown in Figure 3.2. To exploit task-level parallelism of the target task set, the initial solution has the number of PEs equal

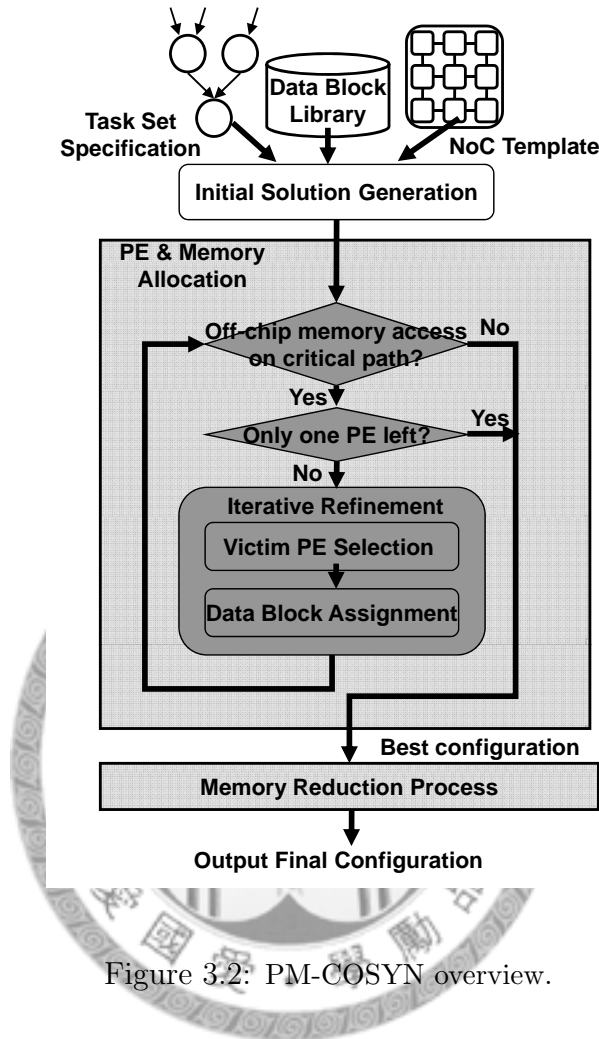


Figure 3.2: PM-COSYN overview.

to the degree of task parallelism and the memory modules allocated to the remaining tiles. In the initial solution, task assignment, data block assignment and tile mapping can be decided by any existing synthesis algorithms, such as [16], [11] and [51]. Starting with the initial solution, PM-COSYN adopts a greedy-based iterative method to refine the system configuration. During the iterative process, which is shown as the *PE&Memory Allocation* step in Figure 3.2, PEs are gradually replaced with memory modules to see if it results in better system performance. This contains two main steps. The first step is *Victim PE Selection* which decides the PE to be replaced, and the assignment of tasks that are original assigned to the replaced PE. After Victim PE Selection, the second step, *Data Block Assignment*, is invoked to decide which data blocks should be assigned to the newly allocated

on-chip memory. This iterative process stops when system critical path does not have accesses to data blocks assigned to the off-chip memory or there is only one PE left on the system. Since in the initial solution, the number of memory module is decided by the degree of task parallelism, it is possible to have more memory modules than required after the PE&Memory Allocation step. Therefore, PM-COSYN performs Memory Reduction Process to reduce the number of memory modules if possible. Next, we will describe the details of the proposed PM-COSYN framework.

3.3.1 PE&Memory Allocation

As described earlier, PE&Memory Allocation is an iterative process, where a PE is replaced with an on-chip memory during each iteration. To improve system performance, the data blocks which are the most critical to the system performance are assigned to a on-chip memory at each iteration. For this reason, as shown in Figure 3.2, an iteration is not started as long as the system critical path has no accesses to data blocks which are assigned to on-chip memories. An iteration is not started if there is only one PE left on the system, either. During each iteration, PE&Memory Allocation performs two major steps, *Victim PE Selection* and *Data Block Assignment*. Victim PE Selection first decides which PE should be replaced by an on-chip memory. Data Block Assignment is then performed to decide which data blocks should be assigned to the newly allocated on-chip memory. Details of Victim PE Selection and Data Block Assignment are described as follows.

3.3.1.1 Victim PE Selection

When selecting a victim PE, many factors need to be taken into account to achieve the best system performance, such as dependency among tasks and the task/data allocation. Due to the complex interplay among these factors, the Victim PE Selection step adopts an exhaustive search method. Each PE $P_i \in P$ in the system is selected as the victim candidate, and its performance impact is evaluated. To reassign the tasks in P_i to other PEs, the Victim PE Selection step examines each task in the decreasing order of the task priority. For each task, all the possible

Victim PE Selection:

Input: $G = \langle V, E \rangle$, $D = \{d_1, \dots, d_k\}$,

$N = \langle T, L \rangle$ and selected PE $P = \{P_1, \dots, P_l\}$

Output: Victim PE P_{vic} and new assignment of tasks

which are originally assigned to P_{vic}

```
1 for each PE  $P_i \in P$ 
2   for each task  $v_j$  assigned to  $P_i$ 
3     for each PE  $P_k \in P - P_i$ 
4       Insert  $v_j$  to  $P_k$ 
5       Evaluate execution time  $T_{(v_j, P_k)}$  of system
6       with  $v_j$  assigned to  $P_k$ 
7       Find the smallest  $T_{(v_j, P_k)}$  for all  $P_k \in P - P_i$ 
8       if only one PE  $P_k$  has the smallest exe. time
9          $P_t(v_j) = P_k$ 
10      else
11        Choose PE  $P_k$  with the most shared data with  $v_j$ 
12         $P_t(v_j) = P_k$ 
13      Evaluate execution time  $T(Vic(P_i))$  of the system
14      taking  $P_i$  as the victim PE
15      Find the smallest  $T(Vic(P_i))$  among all  $P_i \in P$ 
16      if only one  $P_i$  has the smallest  $T(Vic(P_i))$ 
17         $P_{vic} = P_i$ 
18      else
19        Randomly choose PE  $P_i$  with the smallest  $T(Vic(P_i))$ 
20         $P_{vic} = P_i$ 
21 Assign all  $v_j$  on  $P_{vic}$  to  $P_t(v_j)$ 
22  $P = P - P_{vic}$ 
```

Figure 3.3: Pseudo code of Victim PE Selection.

assignments are evaluated. So as shown in Figure 3.3 which lists the pseudo code of the Victim PE Selection step, there are for loops (line 1, 2 and 3) to perform the exhaustive search. The solution that causes less performance degradation is selected as the output of the Victim PE Selection.

Victim PE Selection is the most time consuming step of PM-COSYN. When moving a task v_j to PE P_k , Victim PE Selection first inserts v_j into P_k (line 4 in Figure 3.3), and then system execution time of moving v_j to P_k is evaluated (line 5). To insert v_j into P_k , a simple insertion sort is performed according to the priorities of the tasks on P_k , and the time complexity is $O(|V| \log |V|)$. To evaluate system execution time, we have to traverse all the nodes and edges on the DFG, and the time complexity is $O(|V| + |E|)$. Task insertion and system evaluation are performed at most $|V||P|^2$ times. So, the computational complexity of Victim PE Selection is $O(|V|^2|P|^2 + |V|P|E| + |V|^2|P|^2 \log |V|)$.

3.3.1.2 Data Block Assignment

To maximize system performance, the Data Block Assignment step decides which data blocks should be assigned to the new on-chip memory module so that the total execution time is minimized. To achieve this, the Data Block Assignment step first identifies the critical path of the task set. Next, we find the data block $d_{critical}$ that is assigned to the off-chip memory and contributes the most data accesses ($num_access \times size(d_{critical})$) to the critical path. $d_{critical}$ is then assigned to the new on-chip memory module, and the critical path of the task set is updated accordingly. The process is repeated until the new on-chip memory module is fully utilized or all data blocks are assigned to the on-chip memory modules. However, it is possible that data blocks on the critical path are all assigned to on-chip memory modules and there is space left on the new on-chip memory module. In such a case, to utilize the on-chip memory space to reduce off-chip traffic, data blocks that are assigned to the off-chip memory and contribute the most data accesses to the system are assigned to the new on-chip memory module. The pseudo code of Data Block Assignment is shown in Figure 3.4.

Data Block Assignment:

Input: $G = \langle V, E \rangle$, $D = \{d_1, \dots, d_k\}$

and $N = \langle T, L \rangle$

Output: Set of data blocks assigned to the
new on-chip memory

- 1 $size_left = Memory_size$
- 2 Identify critical path of the task set in current configuration
- 3 while $size_left > 0$
 - and $\exists d_i \in D$ that is assigned to off-chip memory
 - and $size(d_i) \leq size_left$
- 4 Identify $d_{critical}$
- 5 Assign $d_{critical}$ to the new on-chip memory
- 6 Update the critical path
- 7 $size_left = size_left - size(d_{critical})$

Figure 3.4: Pseudo code of Data Block Assignment.

3.3.2 Memory Reduction Process

Since in the initial solution, the number of memory module is decided by the degree of task parallelism and PM-COSYN replaces a PE by an on-chip memory at each iteration, it is possible to have more memory module than required after the PE&Memory Allocation step. To reduce area cost, Memory Reduction Process is adopted to reduce the number of on-chip memory if possible. The process removes an on-chip memory at a time, and evaluates the performance of systems with reduced number of on-chip memory. We use $T(RM(i))$ to denote system execution time of i removed on-chip memories. Given a performance degradation threshold value th , system with the largest number of removed on-chip memory and have performance degradation smaller than th (e.g. $(T(RM(i))/T(RM(0)) - 1 < th)$) is chosen as the final result. To maximize the performance of systems with reduced number of on-chip memories, whenever an on-chip memory is removed, all data blocks assigned to on-chip memories are first moved to off-chip memory, and data block assignment for each of the remaining on-chip memory is decided by the Data Block Assignment process described in Section 3.3.1.2.

3.4 Experimental Results

In this section, we evaluate the effectiveness of PM-COSYN and discuss its experimental results. The experimental setup is described in Section 3.4.1. Analysis of PM-COSYN is performed in Section 3.4.2. We also compare the proposed PM-COSYN algorithm with a Simulated-Annealing optimizer, and the results are presented in Section 3.4.3.

3.4.1 Experimental Setup

To analyze the proposed algorithm, we apply PM-COSYN to two sets of benchmarks, synthetic task sets and real-world applications. The synthetic task sets are generated by the graph generator TGFF [17], which is a parameterizable graph generator. We generate random DFGs and random data block libraries for

Table 3.1: Task set properties.

	Task Set ID	Parallelism Degree	trans_bit/task_cycle	Memory Footprint (in bits)	Initial PE/MEM
Synthetic Task Sets	t0	10	10.468801	276800	9/0
	t1	10	1.20632	279100	9/0
	t2	10	0.094262	276800	9/0
	t3	10	0.051115	276800	9/0
Real-World Applications	Consumer+Telecom	13	0.95139	9018000	9/0
	Mpeg2_Enc+Mpeg2_Dec	2	0.03444	5455900	9/0

every task set. Each of the task sets varies in its data access to task execution ratio. We use the average number of bits transferred per task cycle (trans_bit/task_cycle) as the measurement metric to quantify the data access to task execution ratio of a task set. In addition to synthetic task sets, we also evaluate PM-COSYN on two real-world application mixes: Mpeg2_Enc+Mpeg2_Dec and Consumer +Telecom. Mpeg2_Enc+Mpeg2_Dec is the mix of Mpeg2 encoder and Mpeg2 decoder [2]. Consumer+Telecom is the mix of consumer and telecom benchmark suites obtained from Embedded System Synthesis Benchmark Suites (E3S) [15]. E3S is a collection of task graphs which are built from the Embedded Microprocessor Benchmark Consortium (EEMBC) benchmark suites [20]. Since existing mobile devices usually supports multimedia and telecommunication applications, we select the mix of consumer and telecom task sets for evaluation. The detailed properties of the task sets evaluated in this thesis are listed in Table 3.1.

As mentioned in Section 3.3, the number of PE and on-chip memory allocated for the initial solution is decided by the parallelism degree of the target task set. The numbers of PE and on-chip memory allocated for the initial solutions of all task sets are also listed in Table 3.1. In our experiments, the initial NoC configuration, including task assignment, data block assignment and tile mapping, is decided by a Simulated-Annealing (SA) engine, which synthesize the NoC configuration such that system performance is maximized. The communication cost among tiles is also considered when generating the initial solution. For the Memory Reduction Process described in Section 3.3.2, we set the performance degradation threshold to 1%.

Table 3.2: Detail configuration of target NoC platform.

Parameter	Values
NoC dimension	3×3
Tile size	$1mm^2$
Router latency	5-cycle
Link width	32-bit
Off-chip memory access latency	64-cycle
On-chip memory module size	32KB
PE buffer size	16KB

The detailed configurations of NoC template we used for experiments are listed in Table 3.2. We use 3×3 NoC for synthetic task sets and real-world application mixes. The capacity of an on-chip memory module is set to 32KB, which is estimated by CACTI5.3 [9] assuming 65nm process and $1mm^2$ tile size. As described in Section 3.2, we assume each tile is either a PE or a memory module, and all PEs are general purpose processors with the same micro-architecture.

To evaluate the system performance of the configuration selected in each PM-COSYN iteration, we perform a simple simulator for the target system. According to the selected configuration and given task priorities, we can schedule the execution of tasks. Once the tasks are scheduled, we can know the schedule of data accesses. Since the position of PEs and on-chip memory modules and the assignment of task and data blocks are known in each configuration, we can know the routing path of a data access is composed by which links in the on-chip network. According to the data access schedule and the links traversed by each data access, we can model the contention of the on-chip network. Therefore, the simulator that we adopt here to estimate the task set execution time is a timing-approximate system simulator that models memory contention and on-chip network contention.

3.4.2 Analysis of PM-COSYN

In this section, we analyze the synthesis results of PM-COSYN. Figure 3.5 shows the number of on-chip memory modules allocated for task sets with various `trans_bit/task_cycle`. In this set of experiments, all the task sets have the same DFG

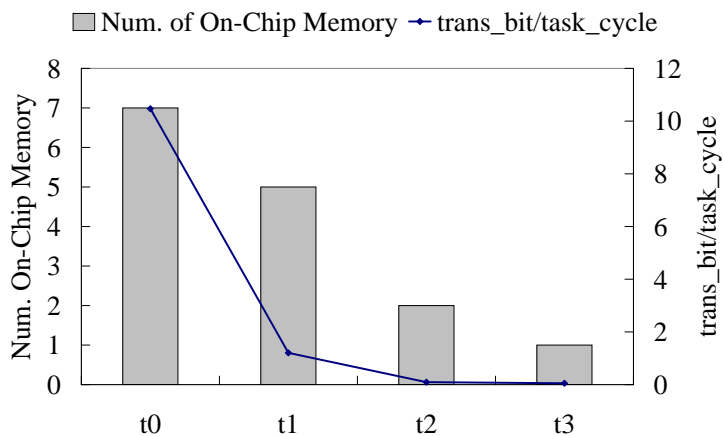
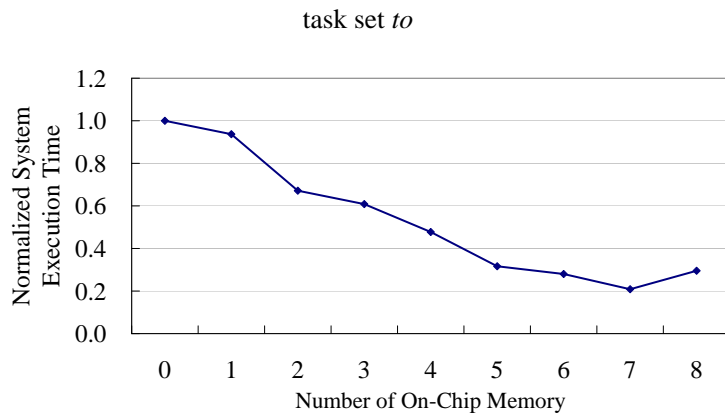


Figure 3.5: Number of on-chip memory allocated for highly parallel task sets with various trans_bit/task_cycle.

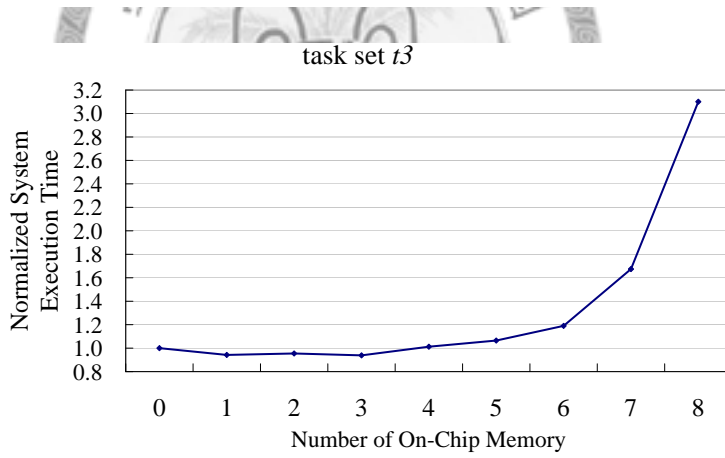
but different data accesses to task execution ratio. The DFG of the task sets used in Figure 3.5 has 51 tasks and maximum parallelism degree of 10, where all chip resources are allocated to PEs when generating the initial solution. From Figure 3.5, we can observe that task sets with large trans_bit/task_cycle also need more on-chip memory modules, while task sets with small trans_bit/task_cycle only need a few on-chip memories. For example, 7 on-chip memory modules are allocated for task set t_0 , which has the highest trans_bit/task_cycle among all tasks. For task set t_3 , which has the smallest trans_bit/task_cycle among the four task sets, only one on-chip memory module is allocated. This shows that task sets with higher data accesses to task execution ratio need more on-chip memories to maximize system performance.

To show the correctness of the solution synthesized by PM-COSYN, we evaluate system performance of NoCs with various number of on-chip memory modules¹. Figure 3.6 shows the results. In this set of experiments, the execution time is nor-

¹We set 9 NoCs with 0 to 8 on-chip memory modules, while the rest of the resources are allocated to PE. Task assignment, data block assignment and tile mapping of PEs and on-chip memory modules are decided by an SA engine for optimizing system performance.



(a)



(b)

Figure 3.6: Normalized system execution time of systems with various number of on-chip memories: (a) task set t_0 , and (b) task set t_3 .

malized to the system without any on-chip memory module. Figure 3.6(a) shows the results of task set $t0$. We can observe that the system with 7 on-chip memory modules achieves the best performance among all configurations, and the result is correspondent to that of PM-COSYN. With 7 on-chip memory modules, the system performance is improved by 79.14% compared to the 9-PE configuration. For task set $t3$, as shown in Figure 3.6(b), the system with 1 on-chip memory module achieves the best performance among all the configurations. This result also agrees with the result of PM-COSYN. Compared to the 9-PE configuration, the system with 1 on-chip memory module improves system performance by 5.8%. Because $t3$ has low $\text{trans_bit}/\text{task_cycle}$, increasing the number of memory module results in performance degradation. For the system with 8 on-chip memory modules, the total execution time is 3.1 times of the 9-PE configuration.

For real-world applications, PM-COSYN allocates 2 on-chip memory modules for Consumer+Telecom, and 1 on-chip memory module for Mpeg2_Enc+Mpeg2_Dec. As the synthetic task sets, we evaluate system performance of the real-world applications in systems with various number of on-chip memory modules, too. The experimental results are shown in Figure 3.7, and the execution time is normalized to the system without any on-chip memory module. Figure 3.7(a) shows the result of Consumer+Telecom. We can observe that the system with 2 on-chip memory modules achieves the best system performance for Consumer+Telecom, which is correspondent to the result of PM-COSYN. The system with 2 on-chip memory modules improves system performance by 13.94% compared to the 9-PE configuration. From Figure 3.7(b), we can see that, the system with 1 on-chip memory module achieves the best performance for Mpeg2_Enc+Mpeg2_Dec. This result is also correspondent to the result of PM-COSYN. Compared to the 9-PE configuration, the system with 1 on-chip memory module improves system performance by 29.34%. Because Mpeg2_Enc+Mpeg2_Dec has low data accesses to task execution ratio, increasing the number of on-chip memory modules degrades the system performance. With 8 on-chip memories, the total execution time is 3.25 times of the 9-PE configuration.

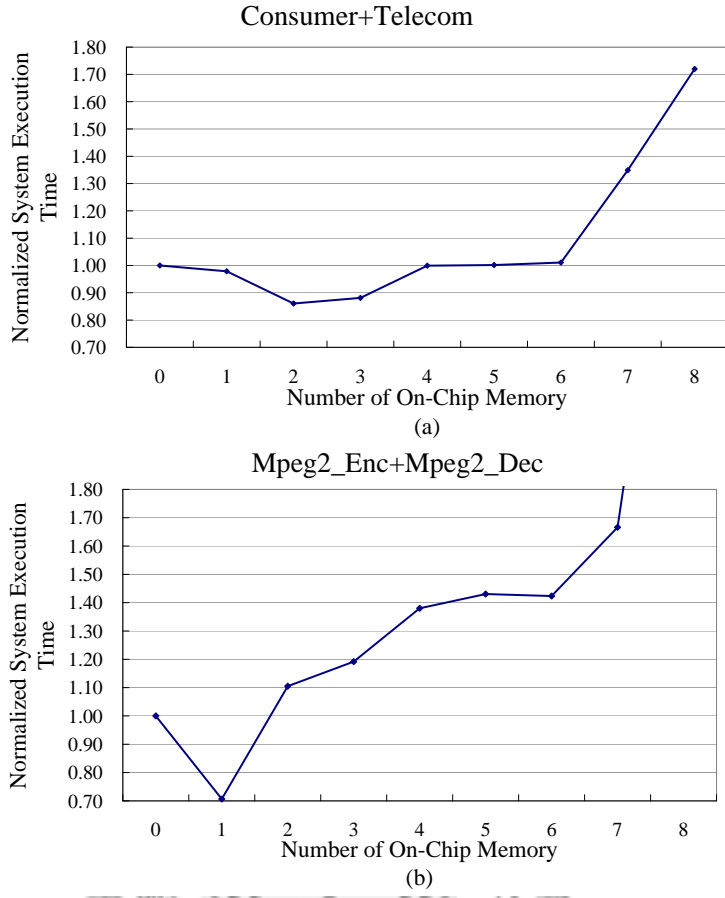


Figure 3.7: Normalized execution time of systems with various number of on-chip memories: (a) Consumer+Telecom, and (b) Mpeg2_Enc+Mpeg2_Dec.

Table 3.3: Comparison of PM-COSYN and PM-SA.

Task Sets	PM-COSYN		PM-SA	
	Execution Time (cycle)	CPU Time (sec)	Execution Time (cycle)	CPU Time (sec)
t0	187237	2.96	192266	741.39
t1	591159	3.05	635475	23754.88
t2	4907203	2.85	4987639	4448.06
t3	8181791	2.87	8676673	6993.70
Consumer+Telecom	78536620	1.17	88664777	20568.68
Mpeg2_Enc+Mpeg2_Dec	156275324	156.02	160162415	176868.15

3.4.3 Comparison with Simulated-Annealing Optimizer

To evaluate PM-COSYN, we also perform a Simulated-Annealing (SA) optimizer called PM-SA² that solves the same PE-memory co-synthesis problem. In this set of experiments, we compare PM-COSYN and PM-SA in terms of solution quality and CPU time as shown in Table 3.3. The solution quality is indicated by the execution time of the task set (in cycles). From Table 3.3, we can observe that PM-COSYN achieves solution quality slightly better than PM-SA with much shorter CPU time. For example, with task set *t2*, PM-COSYN uses only 0.07% of PM-SA CPU time, and the solution quality is 1.62% better than PM-SA. We also perform another set of experiments that see what is the solution quality achieved by PM-SA when it has the CPU time same as PM-COSYN. Table 3.4 shows the results. We can see that, with the CPU time same as the PM-COSYN, PM-SA synthesize solutions with qualities that is up to 32.14% worse than PM-COSYN. Note that with task set *t2* and *t3*, the solution quality of PM-SA is only slightly worse than PM-COSYN. As mentioned in Figure 3.5, *t2* and *t3* are the task sets with low trans_bit/task_cycle, which means *t2* and *t3* are computation-intensive and need more PEs than on-chip memories. As shown in Figure 3.5, *t2* and *t3* need only 2 and 1 on-chip memories, respectively. Therefore, starting with the initial solution that has all PEs and no on-chip memories, PM-SA is easier to achieve good solution quality in a short time when synthesizing solutions for computation-intensive task sets.

Table 3.4: Comparison of PM-COSYN and PM-SA.

Task Sets	Normalized Task Set Execution Time	Improvement over PM-SA
t0	72.66%	17.34%
t1	67.86%	32.14%
t2	98.39%	1.61%
t3	90.91%	9.09%

²PM-SA was optimized by carefully selecting parameters such as number of moves per temperature, cooling schedule, etc.

3.4.4 Scalability Issue

As analyzed in Section 3.3, the program execution time of PM-COSYN grows with with number of tasks. To see how the program execution scales with task set size, we test PM-COSYN on 4×4 NoC (with 16 tiles in total) with task sets having 101, 149, 201 and 249 tasks. The results are shown in Figure 3.8. We can see the program execution time grows rapidly with task set size. With 101 tasks, only 93 seconds are used to solve the problem. With 249 tasks, 1725 seconds (about 29 minutes) are used to solve the case. However, the current scale of the MPSoCs should not be over 249 tasks, and solving the problem in no more than half an hour is still an acceptable result.

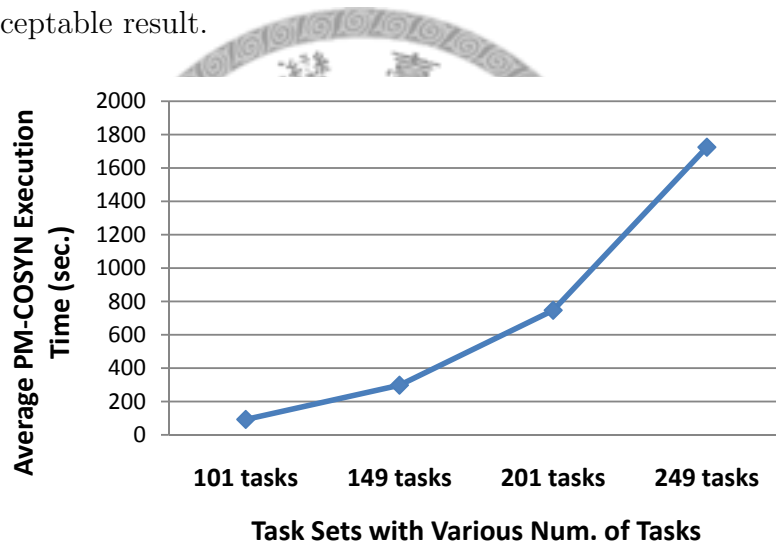


Figure 3.8: How PM-COSYN program execution time scales with task set set size.

Chapter 4

Distributed Memory Interface Synthesis for MPSoCs with Stacked DRAM

In this chapter, we present the distributed memory interface synthesis framework for MPSoCs with stacked DRAMs. We first present the introduction and motivation of the synthesis problem. Then, we give an overview of the 3D integration technology. Next, we define the target 3D MPSoC architecture, and then present the formal problem formulation of the synthesis algorithm. The proposed Simulated-Annealing based algorithm is then presented. Experimental results and analysis are discussed after.

4.1 Introduction and Motivation

To mitigate interconnect-related problems in deep submicron, the emerging 3D integration technology has been proposed. 3D integration technology uses the low-latency and high bandwidth Through Silicon Via (TSV) to stack multiple active device layers together in the third dimension [69]. 3D technology has the property of heterogeneous integration [14], which allows devices with different process technologies, such as high-speed COMS with high-density DRAM, to be integrated on the same chip. Therefore, with the emerging 3D integration technology, it is natural to stack DRAM directly on top of a processor to attack the memory wall issue [30, 39, 42, 67, 69]. Compared to a traditional 2D CPU-DRAM connection design that accesses DRAM through off-chip bus, 3D-stacked DRAM shortens the

access latency. In 3D-stacked DRAM, TSVs are used as the vertical buses to connect PEs and the stacked DRAM. Due to the high density of TSVs, large DRAM bandwidth can be provided, e.g. over three hundreds 1K-bit CPU-DRAM buses on a 1cm^2 chip [39]. To effectively utilize the large DRAM bandwidth, each PE with high DRAM bandwidth demand can have a local DRAM memory controller (DMC), so that the PE can directly access the DRAM stacked on top of it [42, 44]. With this idea, we can see that a MPSoCs with stacked DRAM can have a distributed memory interface where several PEs have their own local DMCs [42].

The baseline architecture of MPSoCs with stacked DRAMs discussed here is shown in Figure 4.1. As shown in Figure 4.1(a), the logic layer is partitioned into regular tiles and different tiles are connected through Network-on-Chip with 2D mesh topology. Each tile has a PE, DMC, a local scratch-pad memory (SPM) module, and a router for connecting the 2D-mesh NoC. One or more DRAM memory layers are stacked on top of the logic layer. Each PE can directly access the DRAM module stacked on-top of it through its local DMC and the vertical bus implemented by TSVs. PEs can address DRAM modules on top of other PEs by transporting the request and data through the horizontal NoC [42]. Figure 4.1(b) shows the detailed view of a DMC. Each DMC has at least three kinds of queues: read, write and command queues. The read queue and write queue take the most transistor budget of a DMC. For a modern DMC design, a read queue or write queue has at least eight slots, and each slot needs about 256-bit capacity. Therefore, the read/write queues occupies transistor budget about 4K-bit SRAM capacity. In addition to multiple queues, a portion of area is occupied by the TSV array for the vertical data bus and control lines to the DRAM module stacked on top of the DMC. As mentioned in [42], with 64-bit data bus and 25-bit control lines, the TSV array occupies about $4 \sim 5\%$ of DMC area. DRAMs stacked on top of the MPSoC are equally partitioned among allocated DMCs. Figure 4.2 shows five different DRAM partitions among five different numbers of DMC allocation. As shown in the figure, the tile position of the DMC for each configuration is fixed. The tile position of each DMC is the one that achieves the shortest data access latency to the DRAM partition that the DMC controls.

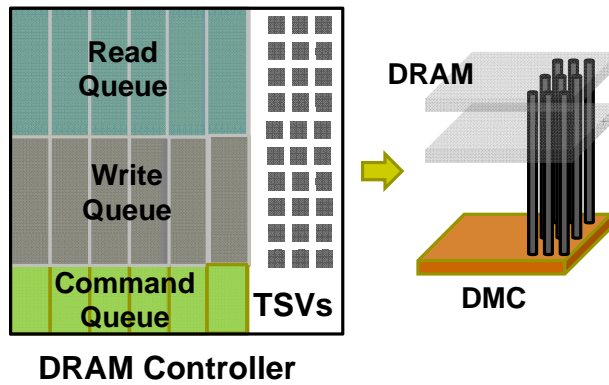
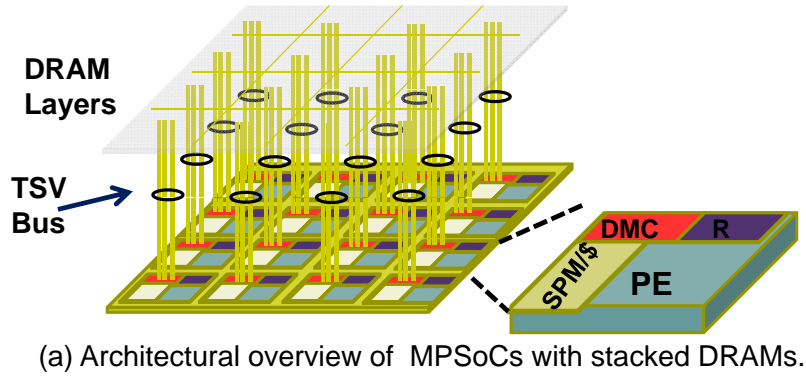


Figure 4.1: Baseline architecture of MPSoCs with stacked DRAMs: (a) Architectural overview of MPSoCs with stacked DRAM, and (b) Detailed architecture of a DRAM controller.

From the discussion of baseline architecture, we can see that, the number of DMCs affects (1) size of on-chip SPM: the transistor budget of the DMC of a tile can be utilized to enlarge the local SPM (as shown in Figure 4.3), (2) DRAM size: since the total DRAM size is fixed and equally partitioned among DMCs, a DMC controls larger DRAM capacity when less number of DMCs are allocated, and (3) Number of TSVs: since the TSVs take area cost, a DMC can have a limited number of data bus width. To have more total DRAM bandwidth, more DMCs should be allocated. Moreover, in addition to area cost, TSVs also have adverse impact on chip design. The manufacturing of TSV needs the steps of drilling, material filling

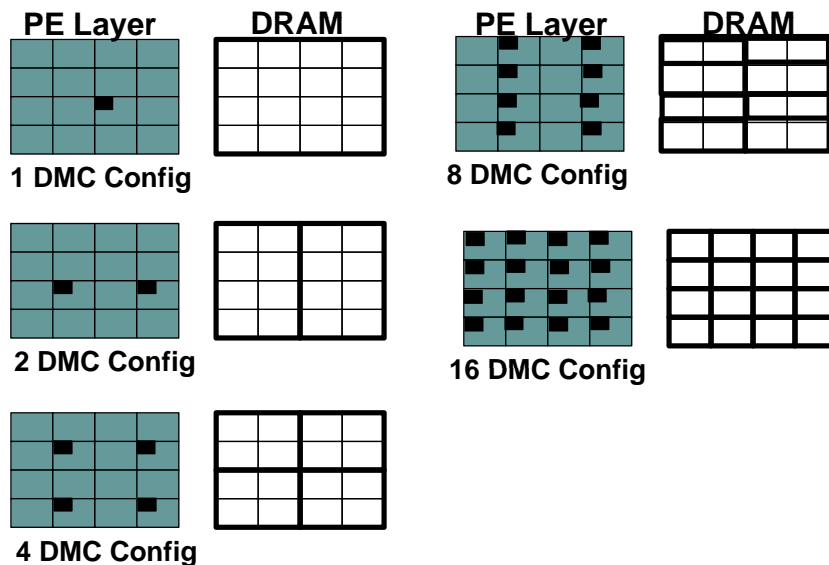


Figure 4.2: 5 different DRAM partitions for different numbers of DMC allocation in a 16-tile MPSoC with stacked DRAM.

and wafer-bonding process [6]. The drilling and wafer-bonding processes expose the chip in high pressure and high temperature environment, which is bad for chip yield [47]. As shown in Figure 4.4, the chip yield degrades as the number of TSVs in a unit chip area increases. Moreover, the material filling process needs to fill the TSV holes with copper or tungsten, which needs extra cost. According to [66], the average cost of making a TSV in wafer-level package is about \$ 0.01 USD. As we can see, although more TSVs provide higher DRAM bandwidth, it also needs higher manufacturing cost and may degrade chip yield. From the above discussion, we can see that the number of DMCs allocated in the system, and the allocation of vertical bus width of each DMC, should be determined carefully so that a design that is balanced between performance and chip manufacturing cost can be achieved.

Therefore, in this thesis, we propose the first distributed memory interface synthesis algorithm for MPSoCs with stacked DRAMs. The proposed algorithm synthesizes the configuration of the distributed memory interface, including the number of allocated DMC and the data bus width of each DMC, according to the requirement of the target application. The goal of the proposed algorithm is to find a

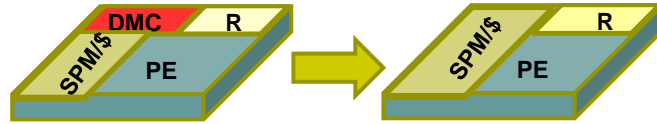


Figure 4.3: Illustration of trading the transistor budget of a DMC to enlarge the local SPM.

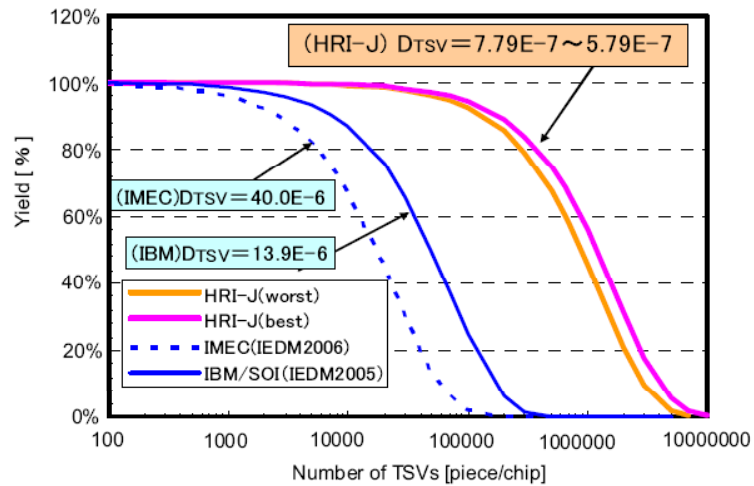


Figure 4.4: TSV connection yield calculated by equation of poisson distribution [47].

distributed memory interface design that meets the performance constraint while the TSV cost is minimized. To tackle this synthesis problem, we proposed an SA-based algorithm. In the following subsections, we first provides a brief overview of 3D integration technology in Section 4.2. The system specifications and formal problem formulation are presented in Section 4.3. The proposed algorithm is described in Section 4.4. The experimental results are discussed in Section 4.5.

4.2 Overview of 3D Integration

There are many candidate technologies for 3D die stacking, but wafer-to-wafer bonding appears to be the mostly discussed technology [7, 8, 26, 39, 54]. As many recent academic studies for 3D stacking technology [30, 39, 41], we also assume this type of technology in the thesis. In wafer-to-wafer bonding, each layer is first fabricated independently as the usual 2D fabrication process. Each wafer is next thinned to only 10 to 100 μm in thickness [7, 39]. This process is called as wafer thinning. Then, the TSVs are etched through the bulk silicon, and thermocompression is adopted to bond each layer together [40]. Currently, there are two commonly used bonding methods, *face-to-face* bonding and *face-to-back* bonding. Face-to-face bonding provides higher via density by processing and depositing 3D vias on top of metal layers as the traditional metal etching technologies [8, 53]. However, it allows only two active layers in a 3D stack. With face-to-back bonding [13, 62], any number of dies can be stacked, but TSVs must be etched through the back side of a die and less via density is possible due to the less resolution of the etching process compared to face-to-face bonding. In this thesis, we assume a face-to-back bonding technology.

Compared to traditional two-dimensional (2D) designs, 3D integrations provides the following advantages for computer architecture design [14]:

1. **Short global interconnects:** According to [69], the vertical distance between two layers is usually in the range of 10 μm to 100 μm . Since the average wire length is reduced, compared to 2D architecture, 3D architecture has better system performance and lower interconnected power consumption.

2. **Heterogeneous integration:** 3D integration allows chips with different technologies to be combined together. For example, in [30], DRAM is combined with processors, and Magnetic Random Access Memory (MRAM) and processor are combined on the same chip in [59].
3. **High memory bandwidth:** Due to high TSV density, 3D stacking enables a memory processor interconnect with very high bandwidth. For example, over three hundreds 1024-bit vertical buses can be implemented within 1cm^2 area [39].

However, compared to the size of a logic cell, the size of TSV is relatively large. For example, the average cell area is $2\mu\text{m}^2$ in 45nm technology, and a TSV occupies up to $10\mu\text{m}^2$ area [32]. Moreover, the defects are easily formed during TSV formation. For example, void may be formed in TSV during the fabrication of TSV, and TSVs of different layers may be misaligned during the bonding process and leads to fail in TSV [27, 36, 61]. Therefore, the allocation of TSVs should be determined carefully.

4.3 System Specifications and Problem Formulation

In this section, we present the models used to describe the target 3D MP-SoC architecture and the formal problem definition of distributed memory interface synthesis problem of MPSoCs with stacked DRAMs.

4.3.1 System Model

Our system consists of an application model and MPSoC architecture model. The application model used for the framework is the same as the one described in Section 3.2.1. As in PM-COSYN, we also assume the priority of tasks are known as a priori. Here, we detail the architectural model and the assumptions of the target MPSoC with stacked DRAM.

The system model specifies the (1) logic layer architecture, (2) the total capacity of the stacked DRAMs, (3) DMC configurations, (4) maximum data bus

width of each DMC, and (5) SPM size and the SPM capacity that a DMC can be traded for. The logic layer is a 2D mesh NoC, therefore, we utilize the same *Architecture Graph* $N = \langle T, L \rangle$ used in PM-COSYN (described in Section 3.2.1) to describe the logic layer of the target MPSoCs with stacked DRAMs. We use variable $Size(DRAM)$ to denote total capacity of the stacked DRAMs. About the DMC configuration, as mentioned in Section 4.1, the DRAM capacity is equally partitioned among the allocated DMCs, therefore, only a set of fixed DMC allocations can be selected as shown in Figure 4.2. We use $DMC = \{dmc_1, dmc_2, \dots, dmc_x\}$ to indicate the set of DMC allocations that can be selected. One and only one of $dmc_i \in DMC$ can be selected. Each $dmc_x \in DMC$ is a binary array of $m \times n$ elements. If the i -th element equals to 1, it indicates a DMC is allocated in the i -th tile when dmc_x is selected. The number of 1s in dmc_x indicates the number of DMCs allocated with the configuration selected. E.g., if we have 4 tiles in the 3D MPSoC platform and the DMC on the second tile position is allocated when only one DMC is allocated, the corresponding $dmc_1 = \{0, 1, 0, 0\}$. As discussed in Section 4.1, due to the area constraint, a DMC can only have a limited vertical bus width. For each DMC, we assume the vertical data bus width can be most B_{max} bits. We use $Size(SPM)$ to denote the baseline capacity of each tile’s SPM, and $Size(DMC)$ to denote the SPM capacity that a DMC module can be traded for. The parameters used to describe the system model are listed in Table 4.1.

4.3.2 Problem Formulation

The goal of the memory system synthesis algorithm is to determine the DMC allocation, and the vertical bus width each allocated DMC for the target application. Since different DMC allocation will affect the data access behavior, e.g. DRAM or SPM data accesses and local or remote accesses, task allocation and data block allocation should be synthesized accordingly so that the performance can be optimized. The goal of the proposed algorithm is to minimize the total number of allocated TSVs so that user-defined performance constraint is met. The formal problem formulation of the 3D MPSoC memory system synthesis problem is defined as the follows.

Table 4.1: Variables used to describe the system model.

Task Graph	
G	$G = \langle V, E \rangle$ the data flow graph of target application
V	the set of tasks in G
E	the set of data transfers among the tasks
D	the set of data blocks accessed in G
3D MPSoC Platform	
X/Y	the X and Y dimension of the logic layer NoC
LW	the link width of the NoC
PE	the set of PEs
DMC	$DMC = \{dmc_1, dmc_2, \dots\}$ the set of DMC allocations can be selected
dmc_i	a binary array of $X \times Y$ components that indicate the allocation of each DMC in each tile
B_{max}	the maximum vertical bus width that can be selected for each allocated DMC
$Size(DRAM)$	total DRAM size
$Size(SPM)$	the baseline SPM capacity of each tile
$Size(DMC)$	the extra SPM capacity that a DMC can be traded for

Given. Dataflow graph represented by a DFG $G = \langle V, E \rangle$, a data block library $D = \{d_1, d_2, \dots, d_k\}$, and MPSoC platform defined by architectural graph $N = \langle T, L \rangle$, DRAM size $Size(DRAM)$, DMC allocations $DMC = \{dmc_1, dmc_2, \dots, dmc_x\}$, the baseline local SPM size $Size(SPM)$, the extra SPM size that a DMC can be traded for ($Size(DMC)$), the maximum data bus width that a DMC can accommodate (V_{max}) and the performance constraint $T_{constraint}$.

Objective. Find a DMC configuration $dmc_i \in DMC$, a vertical bus configuration $B' = \{b_0, b_1, \dots, b_{X \times Y}\}$, the task assignment function ϕ and the data block assignment function ω such that task set execution time T_{exe} is no more than $T_{constraint}$, and the number of TSVs allocated in the system (N_{TSV}) is minimized.

The four steps of DMC Allocation, Vertical Bus Allocation, Task Assignment and Data Assignment for determining dmc_i , V' , ϕ and ω are defined as follows.

- **DMC Allocation:** Determine the DMC configuration for the target application. Only one of the DMC configuration $dmc_i \in DMC$ can be selected.

- **Vertical Bus Allocation:** Determine the vertical bus width for accessing the stacked DRAM module for the allocated DMCs. B' is an array with $X \times Y$ elements. Each $b_i \in B'$ represents the vertical bus width of the DMC allocated in the i th tile. If no DMC is allocated in the i th tile, b_i is zero. Since modern computer systems are byte addressing, we assume that vertical bus width b_i should be power of two and is no smaller than eight and no larger than B_{max} ($8 \leq b_i \leq B_{max}$).
- **Task Assignment:** Assign each task node $v_i \in V$ to one of the PE in tile $t_i \in T$. A task v_i can be assigned to only one PE, and we use the function $\phi : V \rightarrow T$ to represent the task assignment process.
- **Data Assignment:** Assign each data block $d_i \in D$ to one of the memory module. A data block can be assigned to either a DRAM module controlled by an allocated DMC, or a SPM module distributed in the tile $t_i \in T$. We use the function $\omega : D \rightarrow DMC' \cup SPM$, where DMC' is the set of allocated DMC, and SPM is the set of SPM. Note that each memory module has limited capacity, therefore, the total size of data blocks assigned to a memory module should be no more than its capacity. That is,

$$\begin{cases} \sum_{\forall d_i \text{ assigned to } SPM_j} \leq \text{Size}(SPM) & \text{if } DMC \text{ is allocated in } t_j \\ \sum_{\forall d_i \text{ assigned to } SPM_j} \leq \text{Size}(SPM) + \text{Size}(DMC) & \text{otherwise} \end{cases} \quad (4.1)$$

$$\begin{cases} \sum_{\forall d_i \text{ assigned to } DRAM_j} \leq \text{Size}(DRAM) / |DMC'| & \text{if } DMC \text{ is allocated in } t_j \\ \sum_{\forall d_i \text{ assigned to } DRAM_j} \leq 0 & \text{otherwise,} \end{cases} \quad (4.2)$$

where SPM_j denotes the SPM module on tile t_j , and $DRAM_j$ is the DRAM controlled by the DMC allocated on tile t_j .

The total number of TSVs including the data bus of each DMC and the control lines used in each DMC. The control lines of each DMC include address bits and control bits, which are fixed for each of the allocated DMC. Assume the number

of control lines of each DMC is $N_{control}$, N_{TSV} can be modeled as

$$N_{TSV} = |DMC'| \times N_{control} + \sum_{1 \leq i \leq X \times Y} b_i \quad (4.3)$$

4.4 Memory System Synthesis Algorithm for MPSoCs with Stacked DRAM

The design space for the memory system synthesis algorithm is huge, and all of the steps in the co-design flow, DMC allocation, Vertical Bus Allocation, Task Assignment and Data Block Assignment actually interplay with one another. Therefore, in this work, we propose a Simulated-Annealing (SA) based memory system synthesis algorithm for MPSoCs with stacked DRAMs. SA is a widely-used non-deterministic algorithm for solving combinatorial optimization problems [34]. Each iteration of SA is composed of three steps. Perturbation results in a new solution through a set of operations. After each perturbation, a feasibility test is required to verify if the solution violates its constraints or not. The quality of the solution is evaluated with a pre-defined cost function. The whole process is repeated until the SA termination condition is met.

The flow of the SA-based algorithm is shown in Figure 4.5. Four steps in the memory system synthesis flow (DMC Allocation, Vertical Bus Allocation, Task Assignment and Data Block Assignment) are treated as perturbation operations. In the SA algorithm, after a perturbation, we schedule the task set according to current system configuration and get the system execution time T_{exe} . The solution is evaluated by the cost function so that we can decide to reject or accept the solution. Once a solution is accepted after the cost evaluation and the solution is not converged, we re-construct the system configuration accordingly. We also call the SA algorithm shown in Figure 4.5 as the *Baseline SA* in this thesis. Some important ingredients of the baseline SA algorithm are defined as follows.

1. **solution space:** The solution space is the combination of DMC allocation, Vertical Bus allocation, Task Assignment and Data Block Assignment. If we define solution space as S , then $S = |DMC| \times |VB| \times |TA| \times |DA|$, where

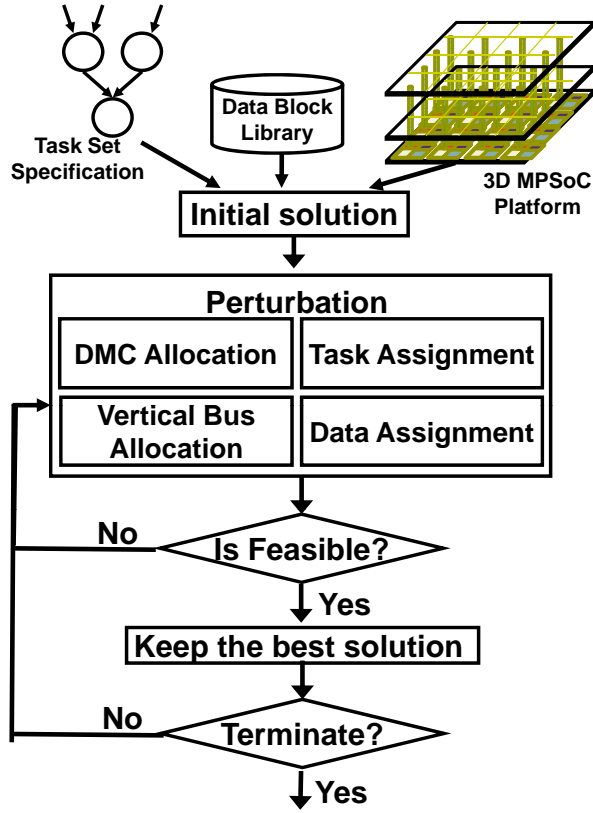


Figure 4.5: Flow of the SA-based memory system synthesis algorithm.

- $|DMC|$: the number of DMC allocations that can be selected.
- $|VB|$: the number of vertical bus width that can be selected for each allocated DMC.
- $|TA|$: the number of task assignment configurations. Assume we have $|V|$ tasks and $|T|$ tiles in the target platform, then, there are $|T|^{|V|}$ kinds of task assignments.
- $|DA|$: the number of data assignment configurations. Assume we have D data blocks and $|T| + |DMC|$ memory modules ($|T|$ SPM modules and $|DMC|$ DRAM modules), then, there are $(|T| + |DMC|)^{|DA|}$ kinds of data assignments.

2. **neighborhood structure**: the neighborhood structure of each perturbation step are described as follows:

- **DMC Allocation:** The perturbation of DMC allocation is to randomly select a DMC configuration $dmc_i \in DMC$. Since different DMC configuration may cause different number of available DRAM modules and the capacity of local SPM, data assignment are re-defined by the data assignment perturbation described later once a data block is assigned to a memory module that no longer exists or do not have enough capacity for it.
 - **Vertical Bus Allocation:** The perturbation of vertical bus allocation is to randomly select a legal data bus width for an allocated DMC that is selected randomly.
 - **Task Assignment:** The perturbation of task assignment is to randomly select a task, and re-assign it to a randomly selected tile.
 - **Data Assignment:** The perturbation of data assignment is to randomly pick a data block, and randomly assign it to an SPM or DRAM module, which has enough capacity for storing the data block.
3. **cost function:** The objective function contains two parts, TSV cost (C_{TSV}) and miss performance constraint penalty ($C_{penalty}$).

$$\cdot C_{TSV} + \cdot C_{penalty}, \quad (4.4)$$

where C_{TSV} is estimated by Eq.(4.3). $C_{penalty}$ is described as following:

$$\begin{cases} C_{penalty} = 0 & \text{if } T_{exe} \leq T_{constraint} \\ C_{penalty} = T_{exe} - T_{constraint} + \epsilon & \text{if } T_{exe} > T_{constraint}, \end{cases} \quad (4.5)$$

where $T_{constraint}$ is the timing constraint of the application, T_{exe} is the current completion time of the application, and ϵ is a constant. Recall that our optimization goal is to minimize the total TSV cost while meeting the user defined performance constraint. In the first case, when the current solution satisfies the specified timing constraints, we concentrate on TSV cost minimization by setting $C_{penalty}$ to zero. In the second case, since the completion time T_{exe} violates the timing constraint $T_{constraint}$, both TSV cost and timing factors should

be considered in searching for the solutions. The $C_{penalty}$ is given more weight as the difference between the timing constraint and the current completion time gets larger. Note that we include ϵ in $C_{penalty}$ to distinguish a feasible solution from an in-feasible solution. The ϵ is an experimental parameter.

Since the perturbation of DMC configuration changes the underlying hardware architecture, it implies that it might require a significant change in task assignment and data assignment as well. For example, if the newly selected DMC configuration has more DMCs allocated in the system, it is very likely the tasks are centralized on certain tiles and cannot explore the advantage of having more DMCs. Therefore, the new solution will be probably rejected by the SA due to its high cost. However, trivially rejecting this new DMC configuration may foreclose possibly attracting DMC configurations. In the example mentioned above, if we re-assign the tasks according to the new hardware configuration, we might be able to find a feasible solution. Therefore, as shown in Figure 4.6, we optimize for the newly selected DMC allocation by performing a low-temperature SA before deciding to accept or reject the new DMC allocation. The low-temperature SA contains task assignment, data block assignment and vertical bus allocation perturbations only. We call this SA engine as the *Low-Temperature SA* method.

4.5 Experimental Results

In this section, we evaluate the proposed distributed memory interface synthesis algorithm. The experimental setup is described in Section 4.5.1. Analysis of the proposed algorithm is performed in Section 4.5.2.

4.5.1 Experimental Setup

To analyze the proposed algorithm, we the algorithm on a set of synthetic benchmarks. The synthetic task sets are generated by the graph generator TGFF [17], which is a parameterizable graph generator. We generate random DFGs and random data block libraries for every task set. Each of the task sets varies in the

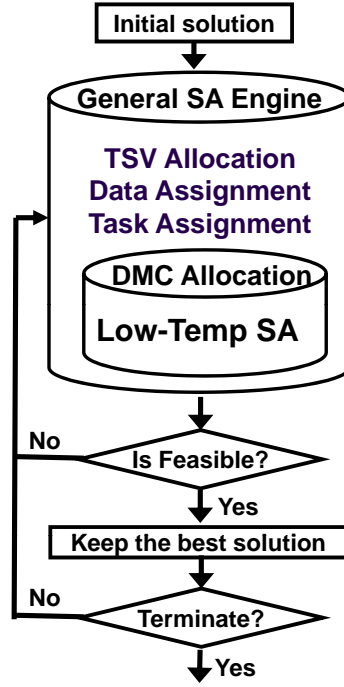


Figure 4.6: Flow of the Low-Temperature SA algorithm.

average number of data accessed in each data transfer. In our experiments, we use a synthetic task graph with 4 different average data transfer per edge, which are 16K-bit, 32K-bit, 64K-bit and 128K-bit. In addition to synthetic task sets, we also evaluate the proposed algorithm on a set of real-world application mixes: Consumer+Telecom. As mentioned in Section 3.4.1, Consumer+Telecom is the mix of consumer and telecom benchmark suites obtained from Embedded System Synthesis Benchmark Suites (E3S) [15]. E3S is a collection of task graphs which are built from the Embedded Microprocessor Benchmark Consortium (EEMBC) benchmark suites [20]. Since existing mobile devices usually supports multimedia and telecommunication applications, we select the mix of consumer and telecom task sets for evaluation.

The detailed configurations of the 3D MPSoC template we used for experiments are listed in Table 4.2. We use 4×4 NoC as the logic layer configuration, and set the on-chip DRAM size to 16Mb, which is large enough to store all the data accessed by the task sets evaluated here. As mentioned in Section 4.3, due to

Table 4.2: Detail configuration of the target 3D MPSoC platform.

Parameter	Values
NoC dimension	4×4
DMC configurations	1, 2, 4, 8, 16 DMCs
Vertical bus width	max 128-bit for each DMC (8, 16, 32, 64, 128 bits)
Router latency	5-cycle
Link width	64-bit
On-chip DRAM size ($Size(DRAM)$)	16Mb
Size of SPM of each PE ($Size(SPM)$)	16Kb
Extra SPM size from de-allocating DMC ($Size(DMC)$)	4Kb
On-chip SPM access latency	1-cycle
On-chip DRAM access latency	12-cycle

the read/write queues, a DMC occupies a significant portion of transistor budget. If the DMC in a tile is not allocated, this resource can be adopted to increase the capacity of local SPM. Assume a DMC with 2 queues, each queue with 4 slots, and each slot has 512-bit storage, the area occupied by DMC can be traded for extra 4K-bit SPM capacity ($Size(DMC) = 4Kb$). According to [42], 64-bit of data bus width and its corresponding control takes about 4.5% of DMC resource. Assume we allow the DMC to have around 10% of area for TSVs, we set the maximum vertical data bus width (B_{max}) of each DMC to 128-bit. Note that, each allocated DMC has fixed 16 vertical control lines, where 11 of them are for address and 5 of them are for DRAM controls, such as RAS and CAS lines [42]. Therefore, we can have maximum of 2304 TSVs in our system. More address lines are required if larger memory addresses space is available.

In our experiments, we set the performance constraint $T_{constraint}$ of each task set to

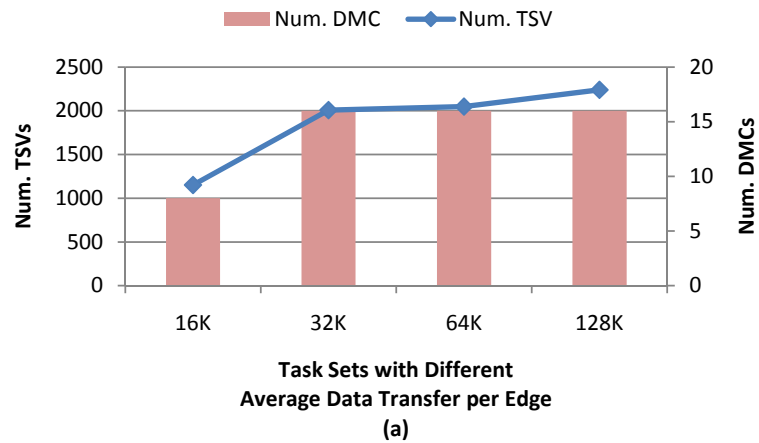
$$T_{constraint} = T_{baseline} \times (1 + Degradation),$$

where $T_{baseline}$ is the task set execution time on the platform with all tiles equipped with a DMC and the vertical data bus is set to max, and $Degradation$ is the percentage of performance degradation we can tolerate. Note that, the performance constraint can be set to other user-specified criteria.

To evaluate the task set execution of each configuration selected at an SA iteration, we perform a simple simulator for the target system as the one we described in Section 3.4.1. Since each DMC may have different vertical bus width and different data access latency in MPSoCs with stacked DRAM, we model how data access latency changes with vertical bus width. Assume a data block d_i with size $Size(d_i)$, DRAM access latency of DL and vertical bus width of b_k , retrieving d_i from stacked DRAM would need $\lceil d_i/b_k \rceil \times DL$ cycles. Same as the simulator used in PM-COSYN, memory contention and on-chip network contention are also modeled here.

4.5.2 Analysis of Experimental Results

In this section, we analyze the synthesis results of the proposed distributed memory interface synthesis algorithm. Figure 4.7(a) shows the number of allocated DMCs and TSVs and Figure 4.7(b) shows the number of TSV reduction compared to the baseline architecture for the synthetic task sets. In this set of experiments, all the task sets have the same task graph (50 tasks and parallelism degree of 16). Different task sets have different average data transfer per edge (ranging from 16K-bit to 128K-bit), and the performance constraint is set to no performance degradation compared to the baseline architecture. From Figure 4.7(a), we can observe that, task sets with higher average data access per edge also have higher DMC and TSV demands. For example, the task set with 16K-bit average data access per edge needs only eight DMCs and total of 1152 TSVs to achieve the task set execution time that the same as the baseline architecture. As shown in Figure 4.7, when comparing to the baseline architecture, 2.78% to 50% of TSVs can be reduced while the task set execution time stays the same. Figure 4.8 shows how number of allocated DMCs and TSVs and TSV reduction change with different criteria of performance constraint. In this set of experiments, the performance constraints are set to 0%, 3% and 9% of performance degradation compared to the baseline architecture. From Figure 4.8, we can observe that more TSV reduction can be achieved when more performance degradation is allowed. With 3% performance degradation, 8.33% to 50% of TSV reductions are achieved. With 6% performance degradation, the 16K-bit test set needs only 4 DMCs. Moreover, up to 12% to 75% of TSVs can be reduced.



TSV Reduction

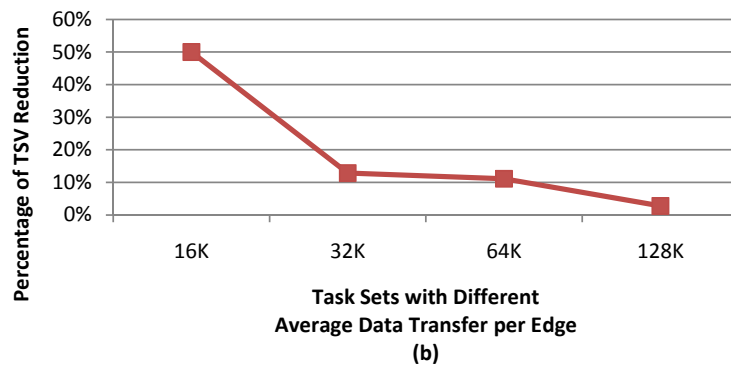


Figure 4.7: Synthesis results for task sets with different average data transfer per edge and performance constraint is set to no performance degradation compared the baseline: (a) Number of allocated DMCs and TSVs, and (b) TSV reduction compared to the baseline architecture.

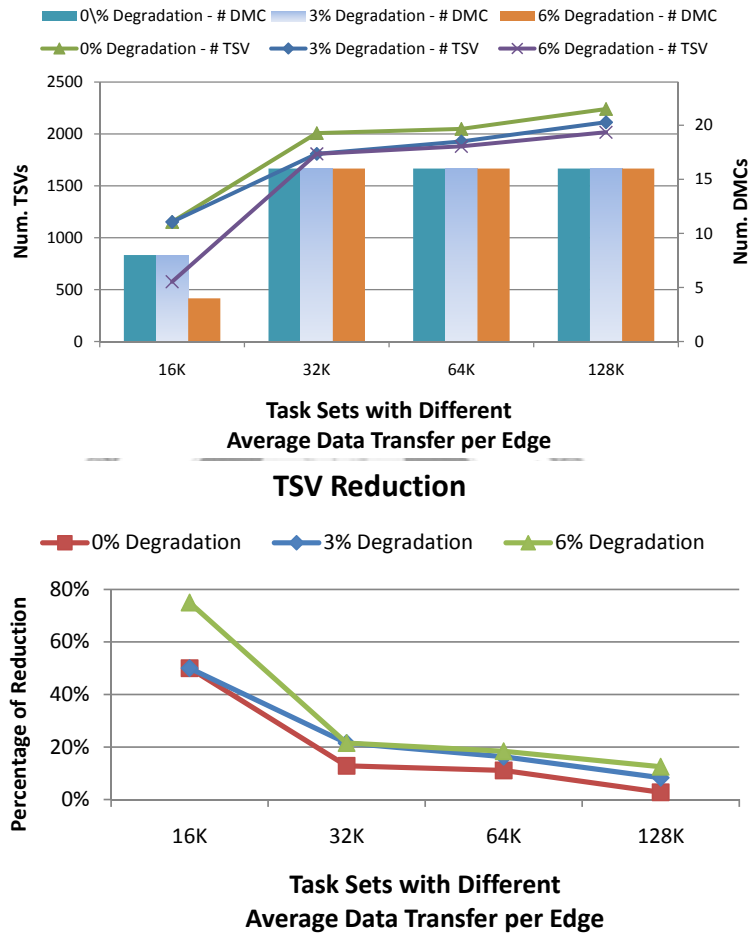


Figure 4.8: Synthesis results for task sets with different average data transfer per edge and performance constraints are set to 0% to 9% performance degradation compared the baseline architecture: (a) Number of allocated DMCs and TSVs, and (b) TSV reduction compared to the baseline architecture.

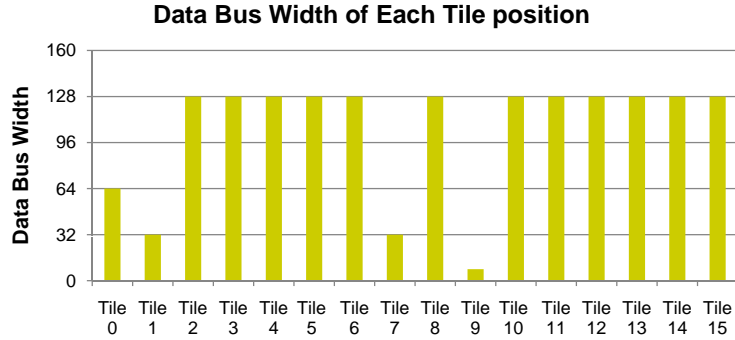


Figure 4.9: Distributed memory interface synthesis results of Consumer+Telecom with performance constraint set to 3% degradation compared to the baseline.

For real-world applications, we test the proposed algorithm on Cosumer+Telecom, which has 42 tasks including applications such as JPEG, auto-correlation, FFT, etc. For Consumer+Telecom, when the performance constraint is set to 3% degradation compared to the baseline architecture, the proposed algorithm synthesized a solution with 16 DMCs and with 27.5% of TSV reductions compared to the baseline. Figure 4.9 shows the results of Consumer+Telecom. Although 16 DMCs are allocated for Consumer+Telecom, we can see that the vertical data bus width of each DMC varies from 8-bit to 128-bit. The results show that TSV can be saved even with all DMCs are allocated since not every DMC need to support the maximum DRAM bandwidth.

Figure 4.10 shows the comparison of the proposed SA method with and without the Low-Temperature SA. We perform this set of experiments on the synthetic task sets, and the performance constraint is set to 3% degradation compared to the baseline. We can observe that, the SA engine without the Low-Temperature SA cannot find a distributed memory interface with reduced TSV cost while the performance constraint is met in all cases. On the other hand, the SA engine with the Low-Temperature SA successfully minimizes TSV cost for all task sets. Table 4.3 shows the average CPU time used to synthesize a solution by SA with and without the Low-Temperature SA. With the average data transfer per edge of 32K-bit case, we can see that the SA with Low-Temperature SA uses only 2.28% more CPU time than the SA without Low-Temperature SA, but achieves up to 21.53% more TSV

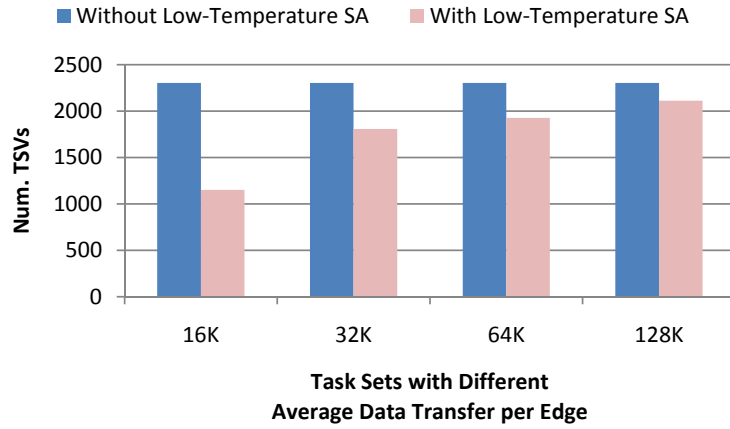


Figure 4.10: Comparison of SA engine with and without Low-Temperature SA.

Table 4.3: CPU time for synthesizing a solution by SA with and without the Low-Temperature SA.

Task Sets	w/ Low-Temperature SA CPU Time(sec)	w/o Low-Temperature SA CPU Time(sec)
16K case	647.919847	384.510294
32K case	223.459636	218.373817
64K case	234.826483	221.427568
128K case	239.346014	180.649925

reduction. This shows the Low-Temperature SA can effectively synthesizes a good solution quality in a short time.

Chapter 5

Concluding Remarks and Future Work

In this chapter, we give the concluding remarks and future work. We have preliminary results of PE and memory co-synthesis framework for traditional 2D NoCs (Chapter 3), and have formulated the PE and memory co-synthesis problem for NoCs with 3D integration (Chapter 4). In the future, we will enhance PM-COSYN for traditional 2D NoCs and complete the co-synthesis framework for 3D MPSoCs.

5.1 PE and Memory Co-Synthesis for Traditional 2D NoCs

In this thesis, we proposed PM-COSYN, the first synthesis algorithm that simultaneously synthesize PE and on-chip memory for application-specific NoCs with the area constraint. The proposed PM-COSYN is a greedy-based iterative algorithm. Starting from an initial solution with all PEs and no on-chip memory, during each iteration, PM-COSYN replaces a PE by a memory and allocates the most critical data blocks to the on-chip memory to improve system performance. The experimental results show that PM-COSYN successfully identifies the correct numbers of PE and memory module that achieves the best system performance. By using a Simulated-Annealing (SA) optimizer as reference, we have shown that PM-COSYN can generate high quality solutions with significantly less computational time for both synthetic task sets and real-world applications. When comparing to the SA optimizer, in our test cases, PM-COSYN uses at most 0.3987% of SA program execution time to synthesize a solution with comparable solution quality.

We plan to extend this research in several directions. First, we plan to utilize PM-COSYN to help characterize how application behaviors affect PE and on-chip memory resource allocation. The characterization can help designers to decide proper MPSoC design at early design stage. Second, since energy consumption is an important issue in MPSoC design, we plan to extend PM-COSYN to consider the energy consumption of the system. We plan to analyze how MPSoC design changes when considering different optimization criteria, e.g. performance vs. energy consumption.

5.2 Memory System Design for MPSoCs with Stacked DRAM

In this thesis, we proposed the first distributed memory interface synthesis algorithm for MPSoCs with stacked DRAM. The proposed algorithm decides the number of DMCs and the vertical bus width of each allocated DMC. The goal of the proposed algorithm is to minimize system TSV cost while the user-defined performance constraint is met. Compared to the baseline architecture that has every PE with a DMC and each DMC with maximum vertical bus width that it can have, the experimental results show that proposed synthesis algorithm achieves up to 50% TSV cost reduction while the performance degradation compared to the baseline is no more than 3%.

In the future, we will extend the proposed algorithm to consider the thermal issue of MPSoCs with stacked DRAM. One major problem of 3D integration is the chip stacking architecture is not good for thermal dissipation. To minimize chip temperature, we can decide how to decide the resource allocation to achieve a balanced design between performance and temperature, e.g. more high-temperature PEs to exploit task parallelism or more low-temperature on-chip memory modules to minimize average data access latency. We can also decide proper task and data assignment so that hot spot in the system can be avoided. Therefore, in the future, we plan to combine the proposed PM-COSYN and distributed memory interface synthesis algorithm to manage the thermal issue of MPSoCs with stacked DRAMs through proper resource allocation and data assignment.

Bibliography

- [1] Moore's law. <http://www.intel.com/technology/mooreslaw/index.htm>.
- [2] Mpeg-2 video. IS standard. I. D. 13818-2, 2001.
- [3] T. Adam, K. Chandy, and J. Dickson. A comparison of list schedules for parallel processing systems. *Communications of the ACM*, 17(12):685–690, December 1974.
- [4] M. Awasthi and R. Balasubramonian. Exploring the design space for 3d clustered architectures. In *Proceedings of the 3rd IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers (P=ac2)*, Yorktown Heights, New York, USA, October 2006.
- [5] 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 *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES '02)*, pages 73–78, Estes Park, Colorado, USA, May 2002.
- [6] P. Benkart, A. Heittmann, H. Huebner, U. Ramacher, A. Kaiser, A. Munding, M. Bschorr, H.-J. Pfeiderer, and E. Kohn. 3d chip stack technology using through-chip interconnects. *IEEE Design and Test of Computers*, 22(6):512–518, November 2005.
- [7] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jian, G. H. Loh, D. McCauley, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '06)*, pages 469–479, Orlando, Florida, USA, December 2006.

- [8] B. Black, D. W. Nelson, C. Webb, and N. Samra. 3d processing technology and its impact on ia32 microprocessors. In *Proceedings of IEEE International Conference on Computer Design (ICCD '04)*, pages 316–318, San Jose, California, USA, October 2004.
- [9] CACTI. Cacti5.3 online available at. <http://www.hpl.hp.com/research/cacti/>.
- [10] G. Chen, F. Li, S. Son, and M. Kandemir. Application mapping for chip multiprocessors. In *Proceedings of the 45th ACM/IEEE Design Automation Conference (DAC '08)*, pages 620–625, Anaheim, California, USA, June 2008.
- [11] Y.-J. Chen, C.-L. Yang, and Y.-S. Chang. An architectural co-synthesis algorithm for energy-aware network-on-chip design. *Journal of Systems Architecture*, 55(5-6):299–309, May 2009.
- [12] J. Cong, J. Wei, and Y. Zhang. A thermal-driven floorplanning algorithm for 3d ics. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '04)*, pages 306–313, San Jose, California, USA, November 2004.
- [13] S. Das, A. Fan, K.-N. Chen, C. Tan, N. Checka, and R. Reif. Technology, performance and computer-aided design of three-dimensional integrated circuits. In *Proceedings of the 2004 International Symposium on Physical Design (ISPD '04)*, pages 108 – 115, Phoenix, Arizona, USA, 2004.
- [14] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Cmineo, A. M. Sule, M. Steer, and P. D. Franzon. Demystifying 3d ics: The pros and cons of going vertical. *IEEE Design and Test of Computers*, 22(6):498–510, 2005.
- [15] R. P. Dick. Embedded system synthesis benchmarks suites (e3s). <http://www.ece.northwestern.edu/~dickrp/e3s/>.
- [16] R. P. Dick and N. K. Jha. Mogac: a multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems. In *Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '97)*, pages 522–529, San Diego, California, USA, November 1997.

- [17] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: Task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101, March 1998.
- [18] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram as a universal memory replacement). In *Proceedings of the 45th ACM/IEEE Design Automation Conference (DAC '08)*, pages 554–559, Anaheim, California, USA, June 2008.
- [19] X. Dong and Y. Xie. System-level cost analysis and design exploration for three-dimensional integrated circuits (3d ics). In *Proceedings of the 2009 Asia and South-Pacific Design Automation Conference (ASPDAC '09)*, pages 234–241, Yokohama, Japan, January 2009.
- [20] EEMBC. Embedded microprocessor benchmark consortium. <http://www.eembc.org/home.php>.
- [21] M. R. Garey and D. S. Jonson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [22] M. Ghosh and H.-H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '07)*, pages 134–145, Chicago, Illinois, USA, December 2007.
- [23] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *ACM SIGARCH Computer Architecture News*, 20(2):278–287, May 1992.
- [24] B. Goplen and S. Spatnekar. Efficient thermal placement of standard cells in 3d ics using a force directed approach. In *Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '03)*, pages 86–89, San Jose, California, USA, November 2003.
- [25] B. Goplen and S. Spatnekar. Placement of 3d ics with thermal and interlayer via considerations. In *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pages 626–631, San Diego, California, June 2007.

- [26] K. W. Guarini, A. W. Topol, M. Leong, R. Yu, L. Shi, M. R. Newport, D. J. Frank, D. V. Singh, G. M. Cohen, S. V. Nitta, D. C. Boyd, P. A. O’Neil, S. L. Tempest, H. B. Pogge, S. Purushothaman, and W. E. Haensch. Electrical integrity of state-of-the-art 0.13 μm soi cmos devices and circuits transferred for three-dimensional (3d) integrated circuit (ic) fabrication. In *Proceedings of the International Conference on Electron Devices Meeting (IEDM ’02)*, pages 943–945, December 2002.
- [27] A.-C. Hsieh, T.-T. Hwang, M.-T. Chang, M.-H. Tsai, C.-M. Tseng, and H.-C. Li. Tsv redundancy: Architecture and design issues in 3d ic. In *Proceedings of 2010 Design, Automation and Test in Europe (DATE ’10)*, pages 166–171, Dresden, Germany, March 2010.
- [28] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architecture under real-time constraints. In *Proceedings of the conference on Design, Automation, and Test in Europe (DATE ’04)*, page 10234, Paris, France, February 2004.
- [29] I. Issenin and N. Dutt. Data reuse driven memory and network-on-chip co-synthesis. In *Proceedings of the International Embedded Systems Symposium (IESS ’09)*, pages 299–312, San Diego, California, USA, June 2007.
- [30] T. Kgil, S. D’Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. Picoserver: Using 3d stacking technology to enable a compact energy efficient chip multiprocessor. *ACM SIGOPS Operating Systems Review*, 40(5):117–128, 2006.
- [31] D. Kim, K. Kim, J.-Y. Kim, S.-J. Lee, and H.-J. Yoo. Solutions for real chip implementation issues of noc and their application to memory-centric noc. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS ’07)*, pages 472–477, Princeton, New Jersey, USA, May 2007.
- [32] D. H. Kim, K. Athikulwongse, and S. K. Lim. A study of through-silicon-via impact on the 3d stacked ic layout. In *Proceedings of the 2009 International*

- Conference on Computer-Aided Design (ICCAD '09)*, pages 674–680, San Jose, California, USA, November 2009.
- [33] S. Kim, C. Im, and S. Ha. Efficient exploration of on-chip bus architectures and memory allocation. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISS '04)*, pages 248 – 253, Stockholm, Sweden, September 2006.
- [34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
- [35] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI (VLSI '02)*, pages 105–112, Pittsburgh, Pennsylvania, USA, April 2002.
- [36] H.-H. S. Lee and K. Chakrabarty. Test challenges for 3d integrated circuits. *IEEE Design and Test of Computers*, 26(5):26–35, Sep./Oct. 2009.
- [37] F. Li, C. Nicopoulos, T. Richardson, and Y. Xie. Design and management of 3d chip multiprocessors using network-in-memory. *ACM SIGARCH Computer Architecture News*, 34(2):130–141, May 2006.
- [38] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari. Bridging the processor-memory performance gap with 3d ic technology. *IEEE Design and Test of Computers*, 22(6):556–564, 2005.
- [39] G. H. Loh. 3d-stacked memory architectures for multi-core processors. In *Proceedings of the 35th International Symposium on Computer Architecture (ISCA '08)*, pages 453–464, Beijing, China, June 2008.
- [40] G. H. Loh, Y. Xie, and B. Black. Processor design in 3d die-stacking technologies. *IEEE Micro Magazine*, 22(6):556–564, 2007.
- [41] G. L. Loi, B. Agarwal, N. Srivastava, S.-C. Lin, and T. Sherwood. A thermally-aware performance analysis of vertically integrated (3d) processor-memory

- hierarchy. In *Proceedings of the 43rd Annual ACM IEEE Design Automation Conference (DAC '06)*, pages 991 – 996, San Francisco, CA, USA, June 2006.
- [42] I. Loi and L. Benini. An efficient distributed memory interface for many-core platform with 3d stacked dram. In *Proceedings of the conference on Design, Automation, and Test in Europe (DATE '10)*, pages 99–104, Dresden, Germany, March 2010.
- [43] M. Lukasiewicz, M. Streubühr, M. Glaß, and C. H. annd J. Teich. Combined system synthesiss and communication architecture exploration for mp-socs. In *Proceedings of the conference on Design, Automation, and Test in Europe (DTAE '09)*, pages 472–477, Nice, France, April 2009.
- [44] A. Marongiu, M. Ruggiero, and L. Benini. Efficient openmp data mapping for multicore platforms with vertically stacked memory. pages 105–110, Dresden, Germany, March 2010.
- [45] S. A. McKee. Reflections on the memory wall. In *Proceedings of the 1st Conference On Computing Frontiers (CF '04)*, pages 162 – 167, Ischia, Italy, 2004.
- [46] P. Mercier, S. Singh, K. Iniewski, B. Moore, and P. O'Shea. Yield and cost modeling for 3d chip stack technologies. In *Proceedings of the IEEE 2006 Custom Intergrated Circuits Conference (CICC '07)*, pages 357–360, San Jose, California, USA, September 2006.
- [47] N. Miyakawa. A 3d prototyping chip based on a wafer-level stacking technology. In *Proceedings of Asia South Pacific Design Automation Conference (ASP-DAC '09)*, pages 416–420, Yokohama, Japan, January 2009.
- [48] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. Exploration of distributed shared memory architectures for noc-based multiprocessors. *Jouranl of Systems Architecture*, 53(10):719–732, October 2007.
- [49] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

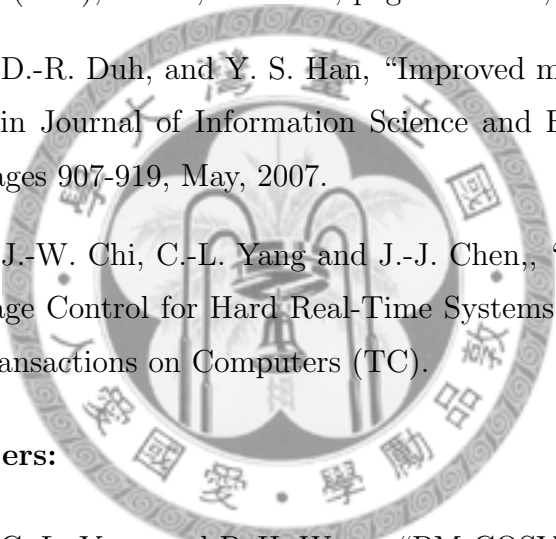
- [50] S. Murali, C. Seiculescu, L. Benini, and G. D. Micheli. Synthesis of networks on chips for 3d systems on chips. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASPDAC '09)*, pages 242–247, Yokohama, Japan, January 2009.
- [51] P. R. Panda, N. D. Dutt, and A. Nicolau. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(3):682–704, 2000.
- [52] S. Pasricha and N. Dutt. Cosmeca: Application-specific co-synthesis of memory and communication architectures for mp soc. In *Proceeding of the conference on Design, Automation, and Test in Europe (DTAE '06)*, pages 700–705, Munich, Germany, March 2006.
- [53] K. Puttaswamy and G. H. Loh. Implementing caches in a 3d technology for high performance processors. In *Proceedings of the 2005 International Conference on Computer Design (ICCD '05)*, pages 525–532, San Jose, California, USA, October 2005.
- [54] R. Reif, K.-N. Chen, and S. Das. Fabrication technologies for three-dimensional integrated circuits. In *Proceedings of International Symposium on Quality Electronic Design (ISQED '02)*, pages 33–37, San Jose, California, USA, March 2002.
- [55] B. Rogers, A. Krishna, G. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: Challenges in and avenues for cmp scaling. In *Proceedings of the International Symposium on Computer Architecture (ISCA '09)*, pages 371–382, Austin, Texas, USA, June 2009.
- [56] C. Seiculescu, S. Murali, L. Benini, and G. D. Micheli. Sunfloor 3d: A tool for networks on chip topology synthesis for 3d systems on chips. In *Proceedings of the Design, Automation and Test in Europe (DATE '09)*, pages 9–14, Nice, France, April 2009.
- [57] M. Shen, L.-R. Zheng, and H. Tenhunen. Cost and performance analysis for mixed-signal system implementation: System-on-chip or system-on-package.

- IEEE Transactions on Electronics Packaging Manufacturing*, 25(4):262–272, October 2002.
- [58] C. Sun, L. Shang, and R. P. Dick. Three-dimensional multiprocessor system-on-chip thermal optimization. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '08)*, pages 117–122, Salzburg, Austria, September/October 2008.
- [59] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel architecture of the 3d stacked mram l2 cache for cmps. In *Proceedings of the IEEE 15th International Symposium on High-Performance Computer Architecture (HPCA '09)*, pages 239–249, Raleigh, North Carolina, USA, March 2009.
- [60] K. Takahashi, H. Terao, Y. Tomita, Y. Yamajo, M. Hoshino, T. S. and T. Morifuji, M. Sunohara, and M. Bonkohara. Current status of research and development for three-dimensional chip stack technology. *The Japan Society of Applied Physics*, 40(4B):3032–3037, 2001.
- [61] A. W. Topol, D. C. L. T. Jr., L. Shi, D. J. Frank, K. Bernstein, S. E. Steen, A. Kumar, G. U. Singco, A. M. Young, K. W. Guarini, and M. Jeong. Three-dimensional integrated circuits. *IBM Journal of Research and Development*, 50(4/5):491–506, Jul./Sep. 2006.
- [62] Y.-F. Tsai, Y. Xie, N. Vijaykrishnan, and M. J. Irwin. Three-dimensional cache design exploration using 3dcacti. In *Proceedings of the 2005 International Conference on Computer Design (ICCD '05)*, pages 519–524, San Jose, California, USA, October 2005.
- [63] S. Udayakumaran, A. Dominguez, and R. Barua. Dynamic allocation for scratch-pad memory using compiler-time decisions. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(2):472–511, 2006.
- [64] B. Vaidyanathan, W. Hung, F. Wang, Y. Xie, V. Narayanan, and M. J. Irwin. Architecting microprocessor components in 3d design space. In *Proceedings*

- of the 20th International Conference on VLSI Design (VLSID '07), pages 103–108, Bangalore, India, January 2007.
- [65] R. Weerasekera, M. Grange, D. Pamunuwa, H. Tenhunen, and L.-R. Zheng. Compact modeling of through-silicon vias (tsvs) in three-dimensional (3-d) integrated circuits. In *Proceedings of the IEEE International Conference on 3D System Integration (3DIC '09)*, pages 1–8, San Francisco, California, USA, September 2009.
- [66] R. Weerasekera, L.-R. Zheng, D. Pamunuwa, and H. Tenhunen. Extending systems-on-chip to the third dimension: Performance, cost and technological tradeoffs. In *Proceedings of the 2007 International Conference on Computer-Aided Design (ICCAD '07)*, pages 212–219, San Jose, California, USA, November 2007.
- [67] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. In *Proceedings of the IEEE 16th International Symposium on High-Performance Computer Architecture (HPCA '10)*, pages 1–12, Bangalore, India, January 2010.
- [68] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, March 1995.
- [69] Y. Xie, G. H. Loh, B. Black, and K. Bernstein. Design space exploration for 3d architectures. *ACM Journal of Emerging Technologies in Computing Systems (JETC)*, 2(2):65–103, 2006.
- [70] X. Zhou, Y. Xu, Y. Du, Y. Zhang, and J. Yang. Thermal management for 3d processors via task scheduling. In *Proceedings of the 37th International Conference on Parallel Processing (ICPP '08)*, pages 115–122, September 2008.

Publication List

Journal Paper:

- 
- [1] Y.-J. Chen, C.-L. Yang and Y.-S. Chang, “An Architectural Co-Synthesis Algorithm for Energy-Aware Network-on-Chip Design,” in *Journal of Systems Architecture (JSA)*, Vol. 5, Issue 5-6, pages 299-309, May, 2009.
 - [2] Y.-J. Chen, D.-R. Duh, and Y. S. Han, “Improved modulo (2^n+1) multiplier for IDEA,” in *Journal of Information Science and Engineering (JISE)*, Vol. 23, no. 3, pages 907-919, May, 2007.
 - [3] Y.-J. Chen, J.-W. Chi, C.-L. Yang and J.-J. Chen., “TACLIC: Timing-Aware Cache Leakage Control for Hard Real-Time Systems,” under second revision for *IEEE Transactions on Computers (TC)*.

Conference Papers:

- [1] Y.-J. Chen, C.-L. Yang and P.-H. Wang, “PM-COSYN: PE and Memory Co-Synthesis for MPSoCs,” in *Proceedings of the conference on Design, Automation, and Test in Europe (DATE 2010)*, Dresden, Germany, pp.1590-1595, March, 2010.
- [2] Y.-J. Lin, Y.-J. Chen, C.-C. Huang, T.-C. Lin, J.-W. Chi, C.-L. Yang, “TunableVP: A Tunable Virtual Platform for Easy SoC Design Space Exploration,” in *Proceedings of IEEE International System-on-Chip Conference (ISOCC)*, Busan, Korea, pp. I-250-I-251, November, 2008.
- [3] J.-W. Chi, Y.-J. Chen, and C.-L. Yang, “Cache Leakage Control Mechanism for Hard Real-Time Systems,” in *Proceedings of International Conference*

on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '07), Salzburg, Austria, pp. 248–256, September - October, 2007.

- [4] W.-H. Hung, Y.-J. Chen, C.-L. Yang, Y.-S. Chang and Alan P. Su, “An Architectural Co-Synthesis Algorithm for Energy-Aware Network-on-Chip Design,” in *Proceedings of 22nd Annual ACM Symposium on Applied Computing (SAC '07)*, Seoul, Korea, pp. 680–684, March, 2005.
- [5] Y.-J. Chen, C.-L. Yang and E.-K. Lin, “Phase-Aware I-Cache Size Synthesis with QoS Consideration,” in *Proceedings of Asia and South Pacific International Conference on Embedded SoCs (ASPICES '05)*, Banagolore, India, July, 2005.

