



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Ph.D. DISSERTATION

Designing Efficient On-chip Networks:  
Mapping, Management, and Routing

온 칩 네트워크 설계: 매핑, 관리, 라우팅

BY

Jinho Lee

February 2016

DEPARTMENT OF ELECTRICAL AND  
COMPUTER ENGINEERING  
COLLEGE OF ENGINEERING  
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Designing Efficient On-chip Networks:  
Mapping, Management, and Routing

온 칩 네트워크 설계: 매핑, 관리, 라우팅

BY

Jinho Lee

February 2016

DEPARTMENT OF ELECTRICAL AND  
COMPUTER ENGINEERING  
COLLEGE OF ENGINEERING  
SEOUL NATIONAL UNIVERSITY

Designing Efficient On-chip Networks: Mapping,  
Management, and Routing

온 칩 네트워크 설계: 매핑, 관리, 라우팅

BY

Jinho Lee

Advisor : Kiyoung Choi

Dissertation submitted to the  
Department of Electrical and Computer Engineering  
in partial fulfillment of the  
requirements for the degree of

Doctor of Philosophy

SEOUL NATIONAL UNIVERSITY

June 2015

Advisory Committee:

Professor	<u>Taewhan Kim</u>	(Chair)
Professor	<u>Kiyoung Choi</u>	(Vice-Chair)
Professor	<u>Jung Ho Ahn</u>	
Doctor	<u>Kyungsu Kang</u>	
Professor	<u>Sri Parameswaran</u>	
Professor	<u>Sungjoo Yoo</u>	

# Abstract

For decades, advance in semiconductor technology has led us to the era of many-core systems. Today's desktop computers already have multi-core processors, and chips with more than a hundred cores are commercially available. As a communication medium for such a large number of cores, network-on-chip (NoC) has emerged out, and now is being used by many researchers and companies. Adopting NoC for a many-core system incurs many problems, and this thesis tries to solve some of them.

The second chapter of this thesis is on mapping and scheduling of tasks on NoC-based CMP architectures. Although mapping on NoC has a number of papers published, our work reveals that selecting communication types between shared memory and message passing can help improve the performance and energy efficiency. Additionally, our framework supports scheduling applications containing backward dependencies with the help of modified modulo scheduling.

Evolving the SoCs through 3D stacking makes us face a number of new problems, and the thermal problem coming from increased power density is one of them. In the third chapter of this thesis, we try to mitigate the hotspot problem using DVFS techniques. Assuming that all the routers as well as cores have capabilities to control voltage and frequency individually, we find voltage-frequency pairs for all cores and routers which yields the best performance within the given thermal constraint.

The fourth and the fifth chapters of this thesis are from a different aspect. In 3D stacking, inter-layer interconnections are implemented using through-silicon vias (TSV). TSVs usually take much more area than normal wires. Furthermore,

they also consume silicon area as well as metal area. For this reason, designers would want to limit the number of TSVs used in their network. To limit the TSV count, there are two options: the first is to reduce the width of each vertical links, and the other is to use fewer vertical links, which results in a partially connected network. We present two routing methodologies for each case.

For the network with reduced bandwidth vertical links, we propose using deflection routing to mitigate the long latency of vertical links. By balancing the vertical traffics properly, the algorithm provides improved latency. Also, a large amount of area and energy reduction can be obtained by the removal of router buffers. For partially connected networks, we introduce a set of routing rules for selecting the vertical links. At the expense of sacrificing some amount of routing freedom, the proposed algorithm removes the virtual channel requirement for avoiding deadlock. As a result, the performance, or energy consumption can be reduced at the designer's choice.

**Keywords:** network-on-chip, scheduling, mapping, thermal management, routing, 3D stacking

**Student Number:** 2011-30251

# Contents

<b>Abstract</b>	<b>i</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Task Mapping and Scheduling . . . . .	2
1.2 Thermal Management . . . . .	3
1.3 Routing for 3D Networks . . . . .	5
<b>Chapter 2 Mapping and Scheduling</b>	<b>9</b>
2.1 Introduction . . . . .	9
2.2 Motivation . . . . .	10
2.3 Background . . . . .	12
2.4 Related Work . . . . .	16
2.5 Platform Description . . . . .	17
2.5.1 Architecture Description . . . . .	17
2.5.2 Energy Model . . . . .	21
2.5.3 Communication Delay Model . . . . .	22
2.6 Problem Formulation . . . . .	23
2.7 Proposed Solution . . . . .	25
2.7.1 Task and Communication Mapping . . . . .	27

2.7.2	Communication Type Optimization . . . . .	31
2.7.3	Design Space Pruning via Pre-evaluation . . . . .	34
2.7.4	Scheduling . . . . .	35
2.8	Experimental Results . . . . .	42
2.8.1	Experiments with Coarse-grained Iterative Modulo Scheduling . . . . .	42
2.8.2	Comparison with Different Mapping Algorithms . . . . .	43
2.8.3	Experiments with Overall Algorithms . . . . .	45
2.8.4	Experiments with Various Local Memory Sizes . . . . .	47
2.8.5	Experiments with Various Placements of Shared Memory . . . . .	48
<b>Chapter 3 Thermal Management</b>		<b>50</b>
3.1	Introduction . . . . .	50
3.2	Background . . . . .	51
3.2.1	Thermal Modeling . . . . .	51
3.2.2	Heterogeneity in Thermal Propagation . . . . .	52
3.3	Motivation and Problem Definition . . . . .	53
3.4	Related Work . . . . .	56
3.5	Orchestrated Voltage-Frequency Assignment . . . . .	56
3.5.1	Individual PI Control Method . . . . .	56
3.5.2	PI Controlled Weighted-Power Budgeting . . . . .	57
3.5.3	Performance/Power Estimation . . . . .	59
3.5.4	Frequency Assignment . . . . .	62
3.5.5	Algorithm Overview . . . . .	64
3.5.6	Stability Conditions for PI Controller . . . . .	65
3.6	Experimental Result . . . . .	66
3.6.1	Experimental Setup . . . . .	66

3.6.2	Overall Algorithm Performance . . . . .	68
3.6.3	Accuracy of the Estimation Model . . . . .	70
3.6.4	Performance of the Frequency Assignment Algorithm . . . . .	70
<b>Chapter 4 Routing for Limited Bandwidth 3D NoC</b>		<b>72</b>
4.1	Introduction . . . . .	72
4.2	Motivation . . . . .	73
4.3	Background . . . . .	74
4.4	Related Work . . . . .	75
4.5	3D Deflection Routing . . . . .	76
4.5.1	Serialized TSV Model . . . . .	76
4.5.2	TSV Link Injection/ejection Scheme . . . . .	78
4.5.3	Deadlock Avoidance . . . . .	80
4.5.4	Livelock Avoidance . . . . .	84
4.5.5	Router Architecture: Putting It All Together . . . . .	86
4.5.6	System Level Consideration . . . . .	87
4.6	Experimental Results . . . . .	89
4.6.1	Experimental Setup . . . . .	89
4.6.2	Results on Synthetic Traffic Patterns . . . . .	91
4.6.3	Results on Realistic Traffic Patterns . . . . .	94
4.6.4	Results on Real Application Benchmarks . . . . .	98
4.6.5	Fairness Issue . . . . .	103
4.6.6	Area Cost Comparison . . . . .	104
<b>Chapter 5 Routing for Partially Connected 3D NoC</b>		<b>106</b>
5.1	Introduction . . . . .	106
5.2	Background . . . . .	107
5.3	Related Work . . . . .	109

5.4	Proposed Algorithm . . . . .	111
5.4.1	Preliminary . . . . .	112
5.4.2	Routing Algorithm for 3-D Stacked Meshes with Regular Partial Vertical Connections . . . . .	115
5.4.3	Routing Algorithm for 3-D Stacked Meshes with Irregular Partial Vertical Connections . . . . .	118
5.4.4	Extension to Heterogeneous Mesh Layers . . . . .	122
5.5	Experimental Results . . . . .	126
5.5.1	Experimental Setup . . . . .	126
5.5.2	Experiments on Synthetic Traffics . . . . .	128
5.5.3	Experiments on Application Benchmarks . . . . .	133
5.5.4	Comparison with Reduced Bandwidth Mesh . . . . .	139
<b>Chapter 6 Conclusion</b>		<b>141</b>
<b>Bibliography</b>		<b>143</b>
<b>초록</b>		<b>163</b>

# List of Figures

Figure 2.1	Motivational example. (a) Example task graph. (b) Example mapping on a 2x2 NoC architecture. (c) Schedule using shared memory communication. (d) Schedule utilizing message passing. . . . .	11
Figure 2.2	Two types of communication. (a) shows shared memory communication and (b) shows message passing. . . . .	13
Figure 2.3	Target many-core SoC platform with a 5x5 2D mesh structure. . . . .	18
Figure 2.4	Target PE architecture. A PE contains a processor, cache, local memory, and an NI. . . . .	20
Figure 2.5	Overall procedure that contains a loop for probabilistic optimization algorithm. . . . .	26
Figure 2.6	Pseudo-code of QEA. . . . .	28
Figure 2.7	Convergence of different encodings. Shift encoding shows the best result. . . . .	30
Figure 2.8	Heuristic algorithm for communication type optimization. . . . .	32
Figure 2.9	Concept of coarse-grained modulo scheduling. . . . .	36
Figure 2.10	Pseudo-code of coarse-grained modulo scheduling. . . . .	37

Figure 2.11	Diagram showing packet scheduling. . . . .	40
Figure 2.12	Experimental results on different mapping algorithms. . .	44
Figure 2.13	Mapping and scheduling results. (a) for energy consumption, (b) for II and (c) for EDP. . . . .	46
Figure 2.14	Experimental result for various local memory sizes. . . .	47
Figure 2.15	Experimental results on various/multiple shared memory locations. . . . .	48
Figure 3.1	An example thermal RC model of a 3D CMP. . . . .	51
Figure 3.2	Normalized application performance numbers for different core/network speeds. . . . .	53
Figure 3.3	The internal structure of a pillar. . . . .	58
Figure 3.4	Hill-climbing algorithm for frequency assignment. . . . .	63
Figure 3.5	Overall structure of the algorithm. . . . .	65
Figure 3.6	Weighted speedup of various approaches. . . . .	68
Figure 3.7	Accuracy of the estimation model. . . . .	70
Figure 3.8	Runtimes of the frequency assignment algorithms. . . . .	71
Figure 4.1	Motivational example. . . . .	73
Figure 4.2	An example of TSV injection/ejection scheme. . . . .	77
Figure 4.3	Flits in TSVs are in deadlock state while flits in 2D layer experience livelock. . . . .	81
Figure 4.4	Performing escape action to avoid deadlock. . . . .	82
Figure 4.5	Proposed router architecture. . . . .	86
Figure 4.6	Pseudo-code of overall routing algorithm . . . . .	88
Figure 4.7	Experimental results under synthetic traffics. Ranges are adjusted for visibility. . . . .	91
Figure 4.8	Application latencies of MCSL on different networks. . .	94

Figure 4.9	Energy consumptions of MCSL on different networks. . . . .	96
Figure 4.10	Energy efficiency of MCSL on different networks. . . . .	97
Figure 4.11	Application latencies on different networks. . . . .	98
Figure 4.12	Energy consumptions on different networks. . . . .	99
Figure 4.13	Energy efficiency on different networks. . . . .	100
Figure 4.14	Average hops on different networks. . . . .	100
Figure 4.15	Application latencies for serialization ratio of 2:1. . . . .	101
Figure 4.16	Energy consumptions for serialization ratio of 2:1. . . . .	102
Figure 4.17	Energy efficiency for serialization ratio of 2:1. . . . .	102
Figure 4.18	Average hops for serialization ratio of 2:1. . . . .	102
Figure 4.19	Fairness comparison with deflection routing and static routing under synthetic traffics. . . . .	103
Figure 5.1	Pseudo-code of the elevator-first algorithm . . . . .	107
Figure 5.2	An example of 3-D stacked mesh topology with regular partial connections in the vertical direction. . . . .	114
Figure 5.3	3-D stacked meshes with random partial connections and the down elevator selection. . . . .	117
Figure 5.4	Illustration of the proof of Lemma 2. . . . .	120
Figure 5.5	Examples of heterogeneous mesh architecture. (a) An architecture that satisfies the topological conditions and (b) the other one that does not. . . . .	124
Figure 5.6	Partially connected mesh architectures for experiments. (a), (b), and (c) have 25%, 50%, and 75% of vertical connections. . . . .	125
Figure 5.7	Comparison on average latencies over various number of vertical connections and traffic patterns. . . . .	129

Figure 5.8	Comparison on aggregate throughputs over various number of vertical connections and traffic patterns. . . . .	130
Figure 5.9	Comparison of EPF over various number of vertical connections and traffic patterns. . . . .	132
Figure 5.10	Comparison of average latency, aggregate throughput, and EPF over various number of vertical connections under uniform random traffic pattern. . . . .	132
Figure 5.11	Application latencies on different routing algorithms. . .	134
Figure 5.12	Energy consumptions on different routing algorithms. .	136
Figure 5.13	Energy-delay products of different routing algorithms. .	138
Figure 5.14	Comparison of partially connected mesh and reduced bandwidth mesh under synthetic traffics. . . . .	139
Figure 5.15	Comparison of partially connected mesh and reduced bandwidth mesh under application traffics. . . . .	139
Figure 5.16	Comparison of partially connected mesh and reduced bandwidth mesh under locally mapped application. . . .	140

# List of Tables

Table 2.1	An Example of Shift Encoding . . . . .	29
Table 2.2	Comparison of Scheduling Methods . . . . .	42
Table 2.3	Test Set Applications . . . . .	45
Table 3.1	Notations Used In the Paper . . . . .	55
Table 3.2	Configuration Parameters . . . . .	66
Table 4.1	System Design Parameters . . . . .	95
Table 4.2	MCSL Application Statistics . . . . .	95
Table 4.3	Area Comparison and Breakdown . . . . .	104
Table 5.1	Network Design Parameters . . . . .	126
Table 5.2	Zero-load Latencies(Cycle) . . . . .	135

# Chapter 1

## Introduction

As semiconductor technology advances, we have been experiencing more and more integration of chips, and System-on-Chip (SoC) has become a big trend in commercialized products since 2000s [1]. In the near future, it is expected that more than hundreds of IPs will be integrated on a single chip. In fact, many companies and researchers are already building many-core architectures [2]. Furthermore, the complexity of applications running on such a chip will be tremendously high, since it will support features like high-resolution, high-quality graphics/videos, high bandwidth communications, immersive computing, etc. Thus communication between cores gets more significant, and therefore, new communication architectures with scalability and large bandwidth are being researched. Among them, bus matrix [3, 4] and Network-on-Chip (NoC) [2, 5, 6] are the most actively researched communication architectures, and especially, NoC is the most viable communication architecture in terms of scalability and bandwidth.

In fact, NoC (Network-on-Chip) has become a research trend, and thus a

number of researches have been carried out in various perspectives to achieve design goals such as scalability, power efficiency, and performance. There are already commercial MP-SoCs that use NoCs as their interconnections [7,8], and there are a huge number of NoC-related publications in diverse areas [6,9–13].

However, designing an NoC is not an easy job, and the challenges range in every level: from the user software to the low-level circuits. This thesis tries to solve some issues, including software mapping/scheduling, thermal management and packet routing. The Following subchapters present the problems and solution overviews.

## 1.1 Task Mapping and Scheduling

In many systems adopting NoC, communications between processor cores are done through a shared memory implemented in the system memory hierarchy [14, 15]. However, this paradigm clearly has limitations due to its latency, energy consumption, and congestion around the shared memory [14]. Furthermore, in the presence of so many computing cores, the overhead for maintaining cache coherence becomes tremendous. As an alternative, many recent architectures support message passing to provide a low-latency, low-energy solution for synchronization and parallel data exchange [2]. However, to support message passing, each processor needs its own local memory to store the messages. When a sizable chunk of data is to be processed without enough room for the data in the processor's local memory, it will be impossible to utilize the message passing for the communication and thus it will require shared memory. Therefore it is better to have both shared memory and message passing.

Many architectures have hardware support for both message passing and shared memory. For example, FLASH [16] Alewife [17], and ASCOMA [18] are

commercially available examples that support hybrid communication. Recently, the Medea framework, a configurable hybrid shared-memory/message-passing architecture, has been proposed [19]. [20] also presents a fabrication of an architecture that supports message passing as well as shared memory. However, no attempt has been made for optimal mapping and scheduling of tasks onto NoC architecture considering optimal selection of communication types among the two alternatives.

The main contributions of this part of thesis (§2) are from [21, 22] and can be stated as follows:

- We propose to map communications to communication types while mapping tasks to the given NoC architecture. We also propose to perform scheduling of the communications and tasks together with the mapping.
- We propose a QEA (Quantum-inspired Evolutionary Algorithm) formulation as well as appropriate encoding for the mapping.
- We consider software pipelining of the given task graph even with backward dependencies when scheduling it on the NoC architecture. For the scheduling problem, we propose a modified coarse-grained modulo scheduling with packet-level communication scheduling integrated.

## 1.2 Thermal Management

The thermal issue is one of the major concerns in architecting CMPs especially for 3D ICs due to the increased thermal density. High chip temperature is known to cause errors to switching transistors, reduce lifetime of the devices, and increase circuit latency as well as leakage power. To prevent such a detrimental impact of high temperature, there has been a significant amount of work on *dynamic thermal management (DTM)*.

Many researches on DTM utilize *DVFS (Dynamic Voltage and Frequency Scaling)* as an important tool to control power consumption. Roughly speaking, DVFS gives linear performance drop with cubic reduction in dynamic power consumption assuming that voltage scales proportional to frequency. Moreover, advances in technology allow fine-grained control of DVFS with small performance overhead. Zhao et al. [23] shows that fine-grained DVFS with overhead of hundreds of nanoseconds is possible. There are also researches on network-level DVFS. [24] proposes DVFS on links to optimize power consumption, and RAFT [25] proposes a router design that allows per-router frequency tuning.

A large portion of work on DTM focuses only on core-level techniques, but it is not sufficient since network-on-chips (NoC) also play an important role in both power and heat dissipation. Considered as an almost necessary backbone component in many-core architectures, many literatures report that NoCs are a major power dissipating source. It is reported that the power portion of NoCs reaches up to 40% of the total system power consumption [26,27]. Also, NoCs have a substantial impact on chip temperature and should be seriously considered as a target for thermal management [28].

Despite the extensive study on DTM in many aspects, none takes advantage of emerging network-level DVFS features to orchestrate cores and networks in a harmonized manner, especially taking into account the application-dependent sensitivity to the speed of cores and routers. Depending on the compute- and memory-intensiveness of the applications, speed control of cores and that of routers have different effects on power and performance. Thus, it is necessary to develop a mechanism to assign appropriate frequency-voltage pairs to both cores and network routers according to application characteristics and current chip temperature. In this paper, we propose a runtime thermal management framework, *THOR* (named after **T**hermal **O**rchestration), that controls cores

and NoC to obtain maximum performance under thermal constraint. The contributions of this part of thesis (§3) from [29] are as follows:

- This is the first work that orchestrates thermal management of cores and network resources at the same time, considering application characteristics.
- We propose the concept of weighted-power budgeting for on-line frequency-voltage assignment of cores and routers.
- We propose a polynomial-time solution for dynamically assigning an optimal voltage/frequency pair to each core/router within a pillar.

### 1.3 Routing for 3D Networks

Among many research areas on NoC, combining 3D stacking technology to build a true 3D NoC with TSVs (not 3D topology mapped to 2D plane) is a natural stream to obtain more integration and higher performance. DimDe [30] has proposed using decomposed crossbar architecture to cope with the complexity due to increased degree of the routers. Wang et al. [31] studied efficient multicasting schemes on 3D NoCs and Rahmani et al published their work on hybrid NoC-bus 3D architectures [32,33]. There is also a 2D NoC implemented using 3D stacking technology that exploits multiple layers to reduce area and energy consumption [34].

Although TSV seems to be a promising solution for inter-layer interconnect in 3D stacking, it suffers from large consumption of silicon area as well as metal area (due to large landing pads). Considering that a low density TSV has a pitch of  $50\ \mu\text{m}$  [35] and high density TSV has a pitch of  $10\ \mu\text{m}$  [36], if a bundle of TSV links has 128-bit width, its area costs  $320\,000\ \mu\text{m}^2$  for low-density TSVs and  $12\,800\ \mu\text{m}^2$  for high density TSVs, just for a single link. It

is a large overhead considering that a traditional 7x7 router consumes about  $100\,000\ \mu\text{m}^2$  at 45 nm process and  $30\,000\ \mu\text{m}^2$  at 22 nm process. Researchers are trying to reduce this pitch, but manufacturers would still want small number of TSVs due to many reasons including misalignment or physical stress problems. There are numerous other techniques for inter-layer connections, but they also encounter the same problem. For example, inductive coupling is one of them and is seriously considered as an alternative to TSVs [37–39]. However, each inductive coil has diameter of few tens of micrometers and the coils should not be placed close to each other because of crosstalk effect. Thus bandwidth between different layers has to be limited. Both Saito et al. [40] and Lee et al. [41] use link serialization to handle this problem.

One way to tackle this problem is TSV serialization, which can greatly reduce the number of TSVs and their footprint at the cost of sacrificing some performance [42]. However, if used in NoCs, it has potential to degrade latency and bandwidth of TSV links significantly and to become the performance bottleneck. The third part of this thesis (§4) targets such an architecture where serialized TSVs are used as vertical connections of 3D NoCs. We devise a way to mitigate such a TSV bottleneck problem by using appropriate routing scheme that allows the flits to fully utilize the TSV links to reduce latency. To be more specific, we use a modified version of deflection routing. Deflection routing has been originally proposed as a low cost, low power mechanism at a minimal performance loss. We focus on the fact that it is naturally a kind of adaptive routing algorithm in a very cheap way. This can be an excellent feature for our target architecture because avoiding congested bottleneck can be more beneficial than seeking a shorter path. We show that if some problems are properly solved, deflection routing can even improve performance compared to static buffered routing while maintaining its original benefits including low cost and

low energy consumption.

Another way of implementing a 3D NoC, is to use partially connected topology instead of serialization [43]. However, the irregularity of such topology makes the routing algorithm vulnerable to deadlock if carelessly designed. The last part of the thesis (§5) proposes an energy-efficient deadlock-free routing algorithm for 3-D NoCs where planar deadlock-free networks are partially connected by vertical links. We focus on mesh topology for the planar networks, which is the most popular one among wide variety of NoC topologies. This comes from its simplicity, scalability, intuitiveness, and easy placement of tiles and wires. Many existing NoC researches have used mesh as their topology [10, 44, 45]. We use dimension ordered (i.e., XY) routing, which was also adopted in [43]. The contributions of this part of thesis are from [46–49] and as follows:

For 3D NoC with limited vertical link bandwidth (§4),

- We propose using deflection routing to solve the TSV bottleneck problem in 3D NoC with TSV serialization.
- We solve a set of new problems including TSV link deflection, deadlock, and livelock to apply deflection routing to the 3D NoC with TSV serialization.
- We provide a thorough evaluation of the proposed architecture in terms of latency, throughput, and power efficiency. For this, we compare the proposed architecture against four others.

For 3D NoC with limited vertical link bandwidth (§5),

- Through a close examination of the layered mesh topology, we prove that the proposed algorithm can be made deadlock-free even without the dedi-

cated VCs for vertical links while maintaining the benefits of the elevator-first routing algorithm: having flexibility that allows random vertical connections and not requiring large lookup tables needed in many topology-agnostic routing algorithms.

- In order to see the effects of VCs on the energy efficiency and performance, the proposed algorithm is evaluated in terms of application latency, energy, and energy-delay product compared to the elevator-first algorithm for 3-D NoCs with different numbers of vertical links connected. The experimental results are measured using NoC performance and power simulator with both synthetic and real benchmark programs.

# Chapter 2

## Mapping and Scheduling

### 2.1 Introduction

Mapping problem is one of the important research problems on Network-on-chips [50] that have been researched over years. However, most researches only consider the location and schedule of the tasks, without considering the type of communication between tasks. On the contrary, our work in this chapter suggests a mapping algorithm which targets a system that provides two communication types: message passing and shared memory. In the architecture that we assumed, both message passing and shared memory communication are done through on-chip network. Under such architecture, compared to message passing, which takes the minimal path from the source to the destination, shared memory communication certainly requires equal or longer network hops since it needs to send data first to the memory and then to the destination. This clearly results in higher latency and energy consumption. Our main contribution in this work is to consider communication mapping and scheduling along

with task mapping and scheduling. With the procedure, one of the possible objective functions—energy consumption, delay (initiation interval), or EDP (Energy-Delay Product)—is minimized.

## 2.2 Motivation

Consider a situation that we are mapping and scheduling a simple task graph comprised of three tasks as shown in Figure 2.1 (a) on a 2x2 NoC architecture shown in Figure 2.1 (b), where the bottom-left tile is occupied by a shared memory. An example of mapping is shown in Figure 2.1 (b). Assuming that all communications are done using the shared memory, the schedule will be like Figure 2.1 (c). Basically, the source task writes data into the shared memory and the destination task reads it<sup>1</sup>. It surely takes long time to travel through the shared memory. Alternatively, we can store the data in the local memory and communicate by message passing to curtail the unnecessary latency. For simplicity, let us assume that the size of the local memory allows only one communication as message passing. Then, among the two communications ( $A \rightarrow B$  and  $B \rightarrow C$ ), we have to choose one that will use message passing. In this example,  $A \rightarrow B$  requires two hops of communication through shared memory and also two hops for message passing.  $B \rightarrow C$  on the other hand, requires three hops through shared memory (one hop for  $B \rightarrow SM$  and two hops for  $SM \rightarrow C$ ) and only one hop by message passing. Thus we chose  $B \rightarrow C$  to use message passing because it will save more latency as well as energy. The resulting schedule is shown in Figure 2.1 (d), to demonstrate that the total execution time can be reduced significantly. This scheme can be applied to save a great amount of energy and execution time. (For example, 59% of energy and 26% of execution

---

<sup>1</sup>For detailed protocol description, refer to § 2.3.

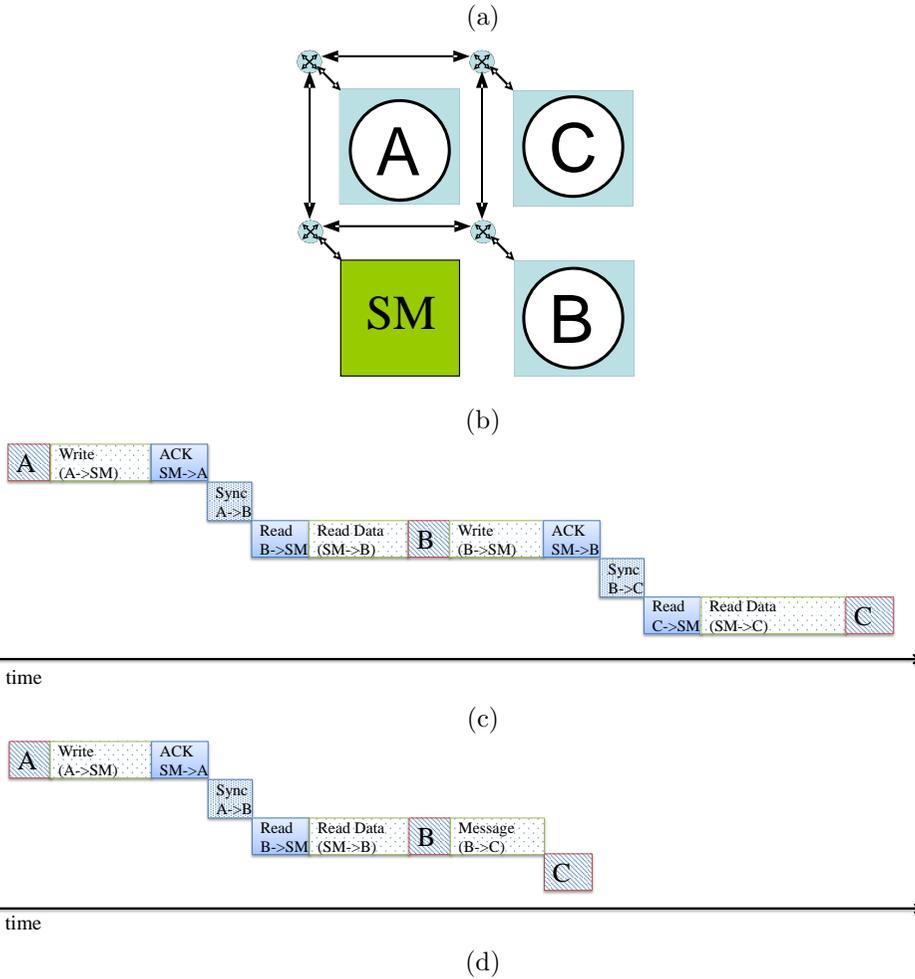
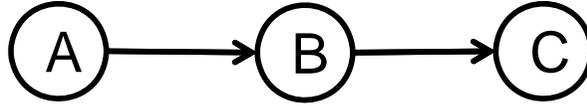


Figure 2.1: Motivational example. (a) Example task graph. (b) Example mapping on a 2x2 NoC architecture. (c) Schedule using shared memory communication. (d) Schedule utilizing message passing.

time for motion-JPEG decoder; see § 2.8.)

## 2.3 Background

In this section, we explain how our scheme works in a big picture and show how each type of communications are done. Let's assume a producer-consumer model where there are two processing elements (PEs) and one global shared memory connected by an on-chip network as shown in Figure 2.2. Each PE has a processor, a cache, a local memory, and a network interface (NI). For simplicity, we assume there is no cache coherence control and point-to-point synchronization message is used instead. When there is a need for synchronization (e.g., multiple writers on the same data), the action is explicitly performed to ensure correctness of the execution. However, our work is still valid for systems with coherence protocols at the cost of increased scheduling effort.

When PE A in Figure 2.2 needs to send data to PE B, it can send through shared memory, or it can take advantage of local memory in each PE to send data through message passing.

If shared memory communication is chosen, PE A (producer) puts data into the shared memory and PE B (consumer) reads them. Since we do not use cache coherent protocol and the data movement is controlled explicitly, the data are immediately transferred to the shared memory, and the memory controller responds with an ACK packet. After receiving ACK packet for writing the data, a synchronization packet is sent via message passing from PE A to PE B to ensure data validity<sup>2</sup>. Upon receiving the synchronization packet, PE B knows that the data have been written to the memory and are safe to be read. Then PE B invalidates the corresponding region in its cache and reads the data. Because

---

<sup>2</sup>Synchronization can also be done using shared memory. However, it will require busy wait on the shared memory resulting in unnecessary latency and contention. For this reason, we decided to always use message passing for synchronization.

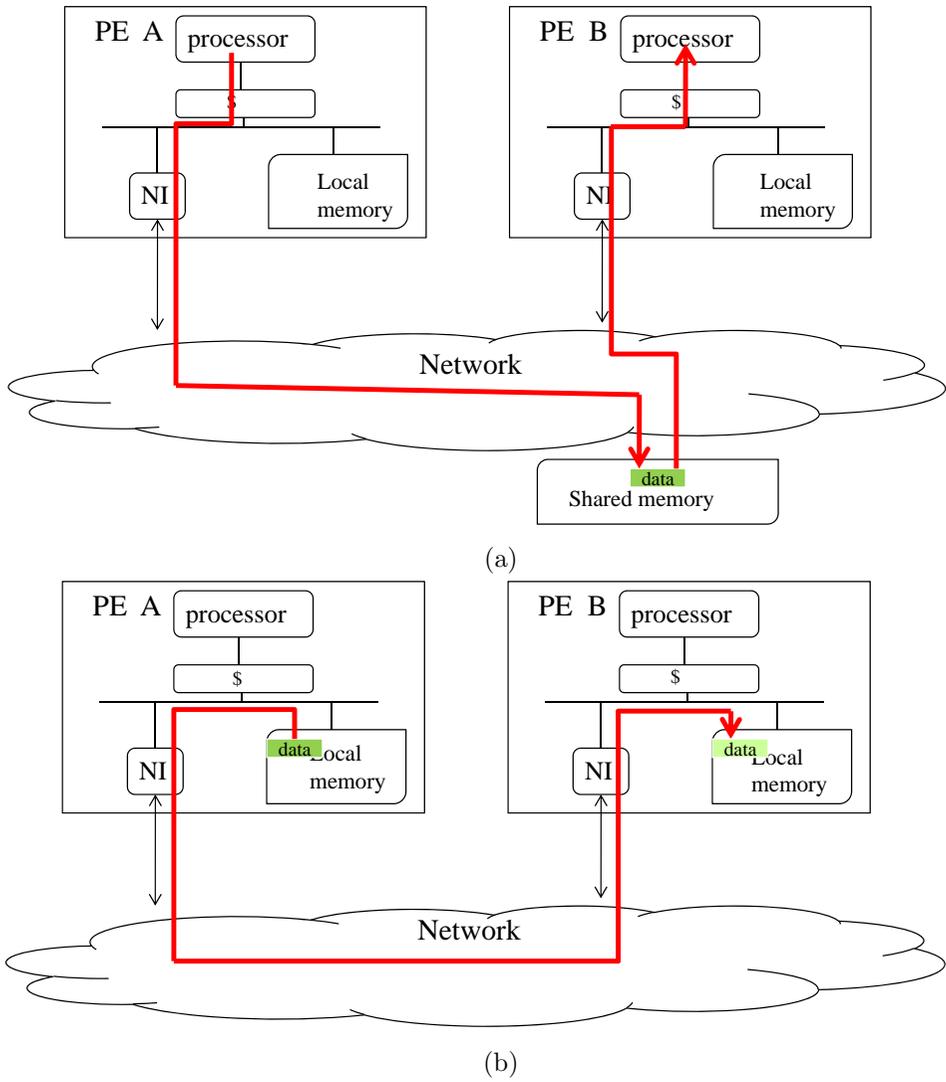


Figure 2.2: Two types of communication. (a) shows shared memory communication and (b) shows message passing.

the region has been just invalidated, cache miss occurs, and read signal from the cache controller is given to the NI. Then the NI sends read packet to the memory tile. The memory controller takes the packet, reads data, and replies back with data packet. Then the NI on the memory tile forwards the data to the cache, and the read operation is completed.

Shared memory communication does not use local memory, since the data is stored in the large shared memory instead of the local memory. One disadvantage of shared memory is long routing path. Data have to make its way to the destination through the memory tile and thus use more links consuming more network energy and requiring more bandwidth and traffic concentration on links around the memory tile. Also, latency and energy consumption from accessing the shared memory itself is another overhead. However, if shared memory communication is used, we can map the application to an architecture with smaller local memory size, or the opportunity to map more tasks on the PE increases with the same size of local memory and thus we can obtain higher CPU utilization and less inter-PE communications. This helps when we are mapping compute-intensive real-time tasks with tight deadlines. Sometimes, when a direct communication path between source and destination is congested, using shared memory for that communication may relieve the congestion due to the traffic splitting.

When communication takes direct message-passing, data are moved from the local memory of the source (PE A) to that of the destination (PE B) through the network. When message passing starts, the processor core in the source PE gives necessary information (start address, size, destination, tag) to the NI. Then the NI reads data from the local memory, packetizes it, and sends the packet through the network. At PE B, the NI writes the data into the local memory. The write address in the local memory is determined based on the tag.

Message passing has an advantage of lower link energy and latency. Since the packet is routed through a shorter path, it usually occupies less number of links, yielding lower link energy consumption and lower latency. Moreover, a passed message is a synchronization object and a data container at the same time. Hence, message passing incurs no additional synchronization overhead. Another advantage comes from not having the latency overhead and energy consumption of large SRAM accesses.

However, when we have limited capacity of local memory and a large volume of data (e.g., one frame of video image) have to be sent, we may not have enough space for buffering the message. Shared memory, on the other hand, is usually much larger than each PE's local memory, making it easy to keep a large volume of data shared by two or more PEs. The problem is that shared memory communication suffers from additional write/read delay and energy which are not needed in message passing. Also, to avoid race condition, cache coherence or other synchronization mechanism is needed. It is another overhead which is needless in message passing because the message itself is a synchronization object. Furthermore, a packet may have to travel a long way in the network to pass through the memory, whereas message passing would possibly take the shortest path directly to the destination for the data transfer. Thus congestion near the shared memory block and large energy consumption due to big memory access can also be serious concerns of shared memory communication.

Considering performance and energy consumption, it would be better to use message passing as much as possible. As stated above, however, we cannot store all data in the local memory when the local memory is not big enough. Thus, we should make a proper decision on the communication type for each data transfer.

It is not an easy problem to do optimal mapping/scheduling of tasks and

communication type mapping considering all these variations and their effects while satisfying the given design constraints. To the best of our knowledge, none of the previous approaches has attempted to solve this problem in its entirety. This paper addresses this issue and tries to solve the problem.

## 2.4 Related Work

There have been other researches on task mapping and/or scheduling onto NoC architectures. Murali and De Micheli [51] and Hu and Marculescu [11] suggest using CWG (Communication Weight Graph) to model target applications as multiple communicating cores and adopting the branch-and-bound algorithm for the mapping. MOCA [52] proposes two-phase method for core mapping and routing. Chou and Marculescu [53] solves contention-aware mapping problem based on ILP. Marcon et al. [54] use CDCM (Communication Dependence and Computation Model) to represent the applications and calculate execution time to get static power consumption. Hu and Marculescu [55] and Xu et al. [56] suggest using CTG (Communication Task Graph) for the mapping and scheduling. However, only a few of these researches consider running times of the tasks [54–56], and even in those cases, they assume the target applications as acyclic graphs which do not contain backward dependencies. This significantly limits the applicability of the approaches because backward dependencies are often found in many applications including video decoders.

There are also numerous researches on mapping SDFG onto multiprocessors [57–59], but they use too simple communication models. For example, the communications between processors are assumed to have constant latencies [57], a simple bus model with bandwidth constraint is used [58], or communication latency is simply ignored [59].

There are also numerous researches on optimization issues related to SPM

(ScratchPad Memory). For example, [60] deals with finding optimal SPM size for each tile, and [61] solves the problem of scratchpad memory partitioning.

[19] and [20] propose architectures supporting both message passing and shared memory. However, they are not proposing any method for optimal mapping of tasks on such architectures. Actually, our approach can be used for application mapping on their architectures with some modification. However, to the best of our knowledge, none of the previous approaches consider exploiting optimization of communication types during the task mapping/scheduling process except for [62]. The work in [62] is the first to reveal that optimizing communication type along with task mapping and scheduling can result in a better solution in terms of application throughput and energy consumption. In the work, communication latency is estimated using a statistical model, which lacks accuracy in timing. In particular, the statistical model estimates network latency under the assumption that packet arrival rates are static. As a result, the estimation does not consider time-varying traffic load of the links. In our experiments, up to 20% of error was observed in total application latency. For accurate timing estimation, we extend the work by integrating communication scheduling together with task scheduling. Also, more extensive experimental results and their analyses are given. They include comparison with other mapping heuristics, analysis of benefits of our mapping/scheduling method in terms of EDP, and analysis of solution quality variation according to the location and number of shared memories.

## **2.5 Platform Description**

### **2.5.1 Architecture Description**

We have designed our homogeneous many-core SoC on a 5x5 2D mesh network structure as shown in Figure 2.3. Each link of the network is 64-bit wide and

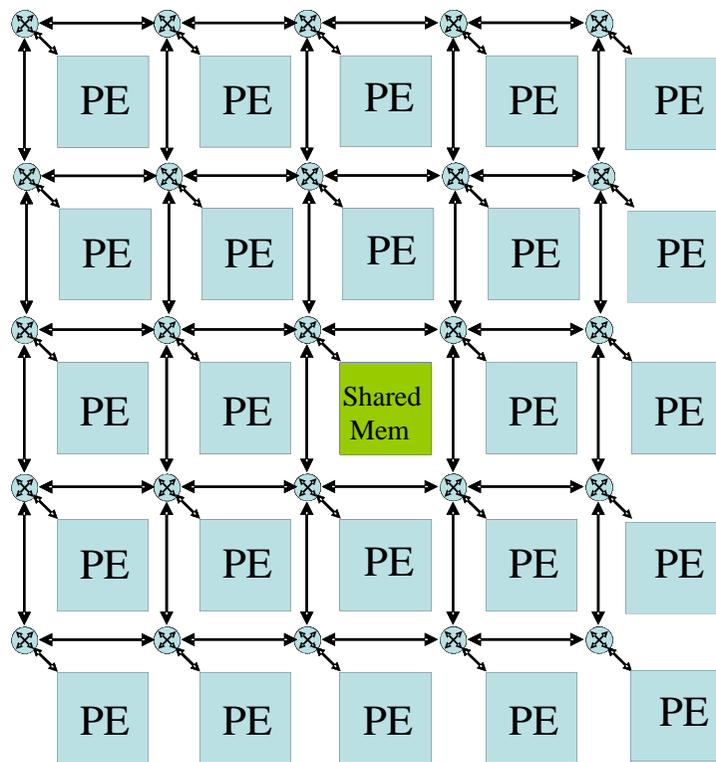


Figure 2.3: Target many-core SoC platform with a 5x5 2D mesh structure.

operates at 400MHz. The first 64 bits of a packet are used as a header which includes information on packet type, size, source, destination, memory address (in shared memory read/write packets), and tag (for message passing). Commonly-used wormhole, XY routing is used for simplicity. However, our work can be applied to a network of any size and any routing method as long as deterministic routing and mesh structure is used. One could also consider clustered architecture to handle a larger system size. Our work can still be applied with the support of partitioning techniques such as the ones in [63,64]. In this paper, however, we focus only on plane mesh architecture.

One of the tiles is occupied by the 64KB (the area obtained by CACTI [65] tool is about the same as that of a processor core [66] and local memory and thus fits within a tile) global shared SRAM and is assigned to a shared region of memory space. It can be used for many purposes besides communication medium between PEs, like storing primary input data or final output. In this study, we place it at the center for efficient sharing, but our algorithm does not limit the location of the shared memory and it can be anywhere among the network tile locations. Currently, our work does not consider any DRAM as a medium for communication, but it can be easily extended for external memories provided that we have a reasonable estimation model for DRAM's delay and energy consumption. The architecture resembles [20], which also has architectural support for both message passing and shared memory, and has center-placed shared memory. The differences are that [20] is comprised of two clusters, each having central shared memory, and shared memory communication does not use the network but uses dedicated wires, which does not look scalable.

The architecture of each PE is shown in Figure 2.4. It has an ARM9 processor running at 200MHz with D-cache, I-cache, local memory, and a network

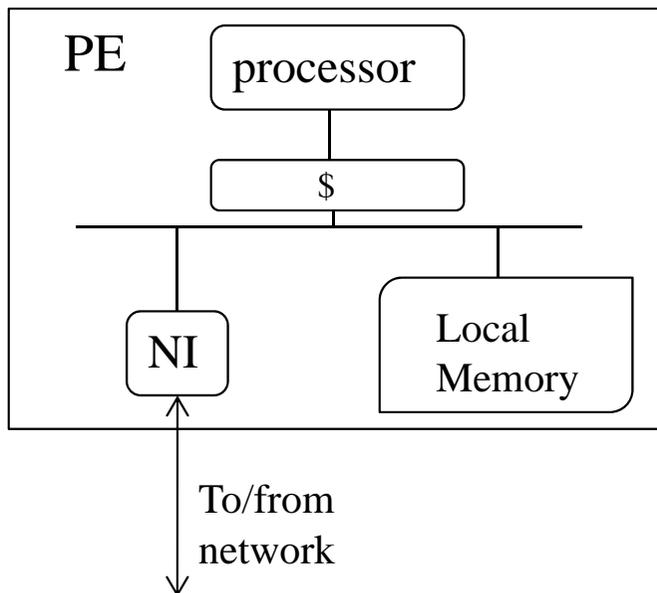


Figure 2.4: Target PE architecture. A PE contains a processor, cache, local memory, and an NI.

interface (NI). The local memory is private. It is mapped to its own private region of the memory address space and can only be accessed by the processor in the same PE. The NI acts as a slave of the processor core and also as a master of the local memory. It handles message passing and shared memory in a DMA-like manner.

The PE has been designed in TLM using Carbon SoC Designer [67], and the network has been also designed in TLM using the ordinary C++ language. For the message-passing, we have implemented eMPI (embedded MPI), a subset of MPI [68], with only three basic primitives—`send()`, `receive()`, and `barrier()`—as in [19] and [69].

## 2.5.2 Energy Model

We calculate energy consumption as

$$E_{total} = E_{NOC} + E_{MEM} \quad (2.1)$$

where  $E_{NOC}$  denotes network energy consumption due to moving data/control packets through the network and  $E_{MEM}$  denotes energy consumptions of the shared memory and local memories.

To calculate  $E_{NOC}$ , we adopt the energy model proposed by Ye et al. [70], which uses bit energy concept to calculate network energy consumption. Network energy consumed for one bit transfer can be calculated by

$$E_{bit} = (n_{hops} + 1) \times E_{Sbit} + (n_{hops}) \times E_{Lbit} \quad (2.2)$$

where  $n_{hops}$  is the number of hops required for the data transfer and therefore  $(n_{hops} + 1)$  represents the number of routers that the bit passes.  $E_{Sbit}$  and  $E_{Lbit}$  denote energy consumed by a router and a link, respectively. Thus the energy consumption for a communication is given by

$$E_{Comm} = E_{bit} \times Data\_Size \quad (2.3)$$

Using these formulas, we can calculate  $E_{NOC}$  as the sum of  $E_{Comm}$ 's over all transactions in the network.  $E_{Sbit}$  and  $E_{Lbit}$  are obtained from Orion 2.0 [71], a network performance and power model.

In addition to energy consumption in the network ( $E_{NOC}$ ), we consider energy consumption in memories ( $E_{MEM}$ ) to see the overall effect of different communication types. CACTI [65] is used to estimate  $E_{MEM}$  consumed in the local memories and the shared memory as a function of read/write accesses to the memories. Energy consumption of the network interface also plays some

part and we also use the Orion model for it. Altogether, these models provide an acceptable estimation for the total energy consumption.

Currently, we do not take into account the energy consumption in the processor cores as in [11, 52–54]. This comes from the reason that the total amount of energy consumed in the processor cores will not change much depending on the task and communication type mapping results for a homogeneous many-core SoC<sup>3</sup>. The portion of network energy in many-core systems is known to be 20-40% of total energy consumption [10, 72, 73].

### 2.5.3 Communication Delay Model

We need to model network latency since it affects run-time of the applications. In this work, we adopt deterministic timing model with explicit scheduling of all the packets. Most of the previous researches model the network latency assuming zero-contention. For example, in [56, 74], they use a simple model given by

$$Delay_{comm} = t_p + \lambda B \quad (2.4)$$

where  $t_p$  is the propagation delay of the header through the path from the source node to the destination node,  $B$  is the number of flits and  $\lambda$  denotes time needed to transmit one flit through one hop distance. This model fits with actual communication delay only when the network traffic is relatively low because it lacks consideration of additional delay caused by network contention.

Instead of the naive model, some approaches schedule each packet to obtain information on communication delay. In [75], a packet scheduling method for guaranteed services in the network is introduced. [54] also uses packet schedul-

---

<sup>3</sup>If power gating or DVFS is used, power consumption on processor cores may be saved. However, we do not consider it in this work because it will make the problem far more complex. We leave it as a future work.

ing method to obtain communication latency, but no explanation is given on how packets are actually scheduled. In [76], an ILP formulation for scheduling and router buffer size optimization is given. We take a similar approach of scheduling packets to obtain communication delays. Since obtaining optimal schedule is impractical for our purpose, we devise a priority based heuristic packet scheduling, which will be explained later in § 2.7.4.

## 2.6 Problem Formulation

Our problem comprises of task mapping, communication type mapping, and scheduling. Thus, taking a *task graph* and an *architecture graph* as inputs, we need to determine on which tile each task should be mapped, which type each communication should take, and when they should be executed such that the total energy consumption, initiation interval (inverse of application throughput), or EDP is minimized, while satisfying various constraints. This section gives a formal definition of the problem.

A task graph  $TG(T, C)$  of an iterative application is a directed graph that may have cycles in it, where each vertex  $t_i \in T$  represents a task, and each directed arc  $c_{i,j} \in C$  represents communication from  $t_i$  to  $t_j$ . Also,  $|T|$  and  $|C|$  represent total number of tasks (nodes) and total number of communications (edges), respectively. An architecture graph  $AG(P, L)$  is a directed graph, where each vertex  $p_i \in P$  represents a PE, and each directed arc  $l_{i,j} \in L$  represents a link from  $p_i$  to  $p_j$  (in a mesh topology, for example, arcs exist between vertices corresponding to adjacent tiles). We choose one of three objectives, initiation interval, energy consumption, or EDP, before running the algorithm. When we choose energy consumption as the objective,  $II_{max}$  is given as a constraint to meet throughput requirement. When we choose throughput as the objective,  $E_{max}$  is given as a constraint and when EDP is the objective,

both  $II_{max}$  and  $E_{max}$  are used as constraints. For the problem formulation, we define the following notations.

- $w(t_i)$ : Worst-case execution time of task  $t_i$ .
- $m_t(t_i)$ : Local memory usage by task  $t_i$ .
- $m_c(c_{i,j})$ : Local memory space needed by  $c_{i,j}$  for message passing.
- $SM, MP$ : Enumeration literals that represent shared memory and message passing, respectively.
- $S_{LM}$ : Size of the local memory in each PE.
- $E_{total}$ : Total energy consumption of the network and memories.
- $E_{max}$ : Upper bound of total energy consumption used as the energy constraint.
- $II$ : Initiation interval obtained by pipelined scheduling (as in § 2.7.4) for a given mapping.
- $II_{max}$ : Upper bound of the initiation interval of the pipelined schedule required to meet the throughput constraint.

Then the problem for given  $TG$  and  $AG$ , is to find a task mapping  $f_t$ <sup>4</sup>

$$f_t : T \rightarrow P$$

and communication type mapping  $f_c$

$$f_c : C \rightarrow SM, MP^{|C|}$$

those minimize  $E_{total}, II$ , or EDP under the constraint of

$$\sum_{t_i: f_t(t_i)=p} m_t(t_i) + \sum_{c_{i,j}: f_t(t_i)=p} m_c(c_{i,j}) + \sum_{c_{i,j}: f_t(t_j)=p} m_c(c_{i,j}) \leq S_{LM}, \forall p \in P \quad (2.5)$$

---

<sup>4</sup>Note that  $f_t$  is not a one-to-one mapping, and multiple tasks are allowed to be mapped to a single tile where their executions will be scheduled.

where  $E_{total}$  is calculated by (2.1) and  $II$  is obtained by scheduling (refer to § 2.7.4). When  $E_{total}$  is chosen as the objective function, an additional constraint for  $II$  is considered by

$$II \leq II_{max} \quad (2.6)$$

Also, when  $II$  is chosen as the objective function, the energy constraint is considered by

$$E_{total} \leq E_{max} \quad (2.7)$$

When EDP is the objective function, both (2.6) and (2.7) have to be considered as constraints. Equation (2.5) is the constraint on the local memory size.

Here we assume that the PEs have uniform local memory size, although the formulation can be easily extended to non-uniform cases. We also assume that the worst-case execution time  $w(t_i)$  (excluding communication delay) of task  $t_i$  on a PE is known a priori. Network delay of message passing between PEs is determined after scheduling and then they are added for the calculation of  $II$ .

## 2.7 Proposed Solution

It is well-known that the mapping problem is intractable [77]. Our problem has an even larger design space than the task mapping only problem because we also need to do scheduling and communication type mapping at the same time. Size of the design space is multiplied by a factor of  $2^{|C|}$  by adding just the communication type mapping problem. For this reason, we have developed our approach based on a probabilistic algorithm.

The overall procedure is shown in Figure 2.5. The procedure consists of a loop formed by the probabilistic algorithm. It starts each iteration by generating a task and communication type mapping through Initial mapping or

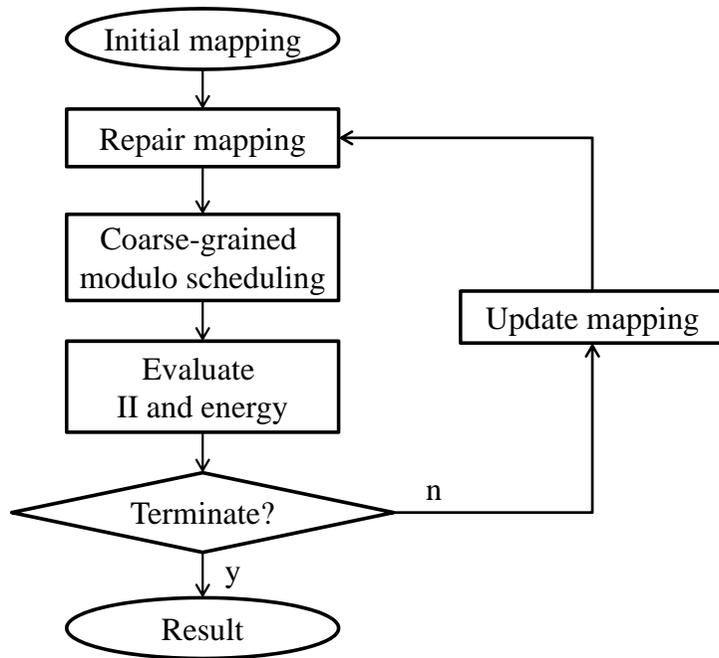


Figure 2.5: Overall procedure that contains a loop for probabilistic optimization algorithm.

Update mapping. Then communication type optimization is applied to remove illegal solutions and improve quality (repair mapping), followed by task/packet scheduling using modified iterative modulo scheduling. In the evaluation phase, solution quality (II, energy or EDP ) is evaluated and constraints are checked. If any of the constraints are violated, the mapping is considered illegal and not used. After these are done, the result is fed back to the probabilistic algorithm for next mapping generation. The following sections explain each step in detail.

### 2.7.1 Task and Communication Mapping

Task mapping problem itself has been researched by a number of research groups, and various approaches have been developed using various methods including exhaustive search, SA (Simulated Annealing), GA (Genetic Algorithm), branch-and-bound, etc. Due to the complexity of the problem, many approaches use probabilistic algorithm such as SA or GA, and we also use such a probabilistic algorithm. In this particular work, we use QEA (Quantum-inspired Evolutionary Algorithm) for task and communication mapping. However, our idea is not limited to QEA only and can be applied to any task mapping algorithm as long as it generates a mapping for tasks and communications.

#### QEA

QEA is an evolutionary algorithm that is known to be efficient compared to other evolutionary algorithms and is becoming popular [78–80]. Figure 2.6 shows the pseudo-code of QEA. It represents the solution space by encoding each solution with a set of Qbits. A Qbit is the smallest unit of information in QEA, which is defined by a pair of numbers  $(\alpha, \beta)$  where  $|\alpha|^2 + |\beta|^2 = 1$ .  $|\alpha|^2$  gives the probability of the Qbit to be found in the ‘0’ state and  $|\beta|^2$  gives the probability of the Qbit to be found in the ‘1’ state. A Qbit may be in the ‘1’

```

1 Procedure QEA
2 begin
3    $t \rightarrow 0$ 
4   while (not termination condition) {
5      $t \leftarrow t + 1$ 
6     for (all  $i$  in population) {
7       make  $P_i(t)$  by observing the states of  $Q_i(t - 1)$ 
8       evaluate  $P_i(t)$ 
9       update  $Q_i(t)$  using Q-gates
10      store the best solutions among  $B_i(t - 1)$  and  $P_i(t)$  into  $B_i(t)$ 
11    }
12    store the best solution among  $B(t)$  to  $b$ 
13    if (global migration condition) migrate globally
14    else if (local migration condition) migrate locally
15  }
16  return  $b$ 
17 end procedure

```

Figure 2.6: Pseudo-code of QEA.

state, in the ‘0’ state, or in a linear superposition of the two and this is how QEA maintains many potential solutions in a compact way, thereby enabling a very fast design space exploration. In this paper, we have chosen QEA because we can easily and efficiently implement our idea of mapping communication types. In QEA, each solution instance is encoded as a binary string. In our problem, the information on the mapping of each task and the type of each communication should be encoded. Since we just need to decide between MP and SM for communication type, this can be easily implemented by encoding the mapping to MP as ‘0’ and that to SM as ‘1’. Methods to encode task mappings for QEA are shown in the next section.

### QEA Encoding for Task Mapping Problem

Whereas encoding for communication type mapping is easy and intuitive, encoding for the mapping of each task has to be done more carefully. A naive approach is to use binary encoding and assign  $\lceil \log_2 |P| \rceil$  Qbits for each task to

represent the mapping to a PE (e.g., “00” for mapping to a PE at tile 0, “01” for tile 1, and so on). This is a very intuitive encoding, but does not give a good result due to the mismatch between encoding patterns and PE locations. In the QEA, we expect a gradual improvement when we modify the code of an individual slightly in the direction toward a better solution. However, if a small change of a code leads to an abrupt movement of the corresponding task to a PE in a long distance, then the algorithm may not be able to guess the right direction.

Another approach is to adopt the *one-hot encoding* used in [79]. However, this is not very nice either, since it requires too many Qbits and does not properly convey the information on distance between tiles. Therefore, we need to devise a new encoding scheme.

*Gray code* seems to satisfy the needed characteristics since only one bit changes between two adjacent values. However, there exists a false relation between numbers that are not adjacent. For example, Gray code for 0 is “00” and Gray code for 3 is “10”. They are close in their Hamming distance, but not in the physical location (refer to Figure 2.3 for their physical locations).

Yet another encoding scheme is shown in Table 2.1. We call it shift encoding because this somewhat resembles logical shift operation. We use this encoding for each dimension, so that the total number of Qbits we need is  $(\text{size}_X-1)+(\text{size}_Y-1)$ . For a 5x5 mesh architecture, it is eight in total. In this

Table 2.1: An Example of Shift Encoding

Y \ X	0	1	2	3	4
0	0000,0000	0001, 0000	0011, 0000	0111, 0000	1111, 0000
1	0000,0001	0001, 0001	0011, 0001	0111, 0001	1111, 0001
2	0000,0011	0001, 0011	0011, 0011	0111, 0011	1111, 0011
3	0000,0111	0001, 0111	0011, 0111	0111, 0111	1111, 0111
4	0000,1111	0001, 1111	0011, 1111	0111, 1111	1111, 1111

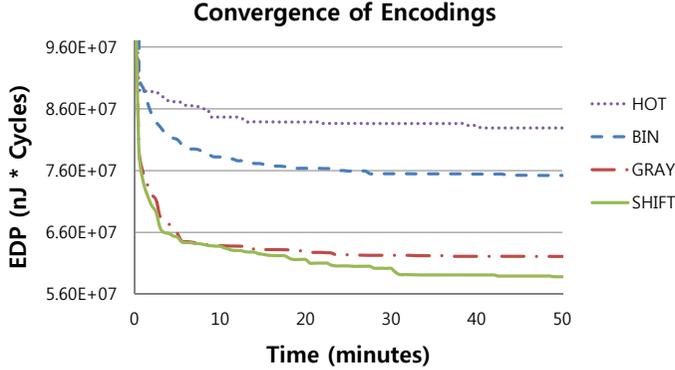


Figure 2.7: Convergence of different encodings. Shift encoding shows the best result.

encoding, Hamming distance and Manhattan distance exactly match and slight modification of encoding gradually changes the solution quality. Thus we can obtain a solution of better quality and good speed of convergence. The shift encoding works better for mesh architecture where the Hamming distance and the Manhattan distance exactly match. Regarding routing algorithm, the encoding scheme is not limited to XY routing but will work equally well for any routing algorithm as long as the routing takes minimal paths. We think it will also be effective for non-minimal routing since the routing tends to return to a minimal path. When the observed binary string does not match with any of the entries in the table (illegal), we simply observe again until we get a valid (legal) string. Alternatively, we could repair the string instead of re-observing it, which may reduce the time spent on multiple observations. However, if we apply repairing, we may lose the direct relation between encoding and its physical location, ruining the purpose of using shift encoding. Moreover, the overhead of re-observing scheme is not very high. For example, 5x5 architecture needs four bits for the encoding and thus 16 binary combinations are possible. Since

five combinations out of the 16 render legal encodings, the initial probability of obtaining a legal observation with a single try is  $5/16 = 31.25\%$ . Note that the probability goes higher as the algorithm proceeds and the Qbits converge. Thus only about two re-observations are needed on average until we obtain a legal one. Because scheduling and evaluation steps take a lot more time, a few re-observations are not really a burden in the running time of the algorithm.

To compare the effects of various encoding schemes, we mapped ten random graphs generated by TGFF [81], a tool for generating dummy task graphs, onto the target architecture described in § 2.5.1. The objective function was set to EDP. The algorithm was run for ten times for each graphs and the average was taken. The results are shown in Figure 2.7. Binary encoding and one-hot encoding show poor quality compared to the others shown in the graph. Both gray and shift encoding show good results, but shift encoding performs better. Thus, for all experiments throughout this work, shift encoding is used.

### 2.7.2 Communication Type Optimization

The design flow works in a way that, after generating mapping solutions, mappings that violate any constraints are thrown away. By doing this, the probabilistic algorithm moves towards generating solutions within constraints. However, we found that it is very slow in its convergence. For this reason, we chose to integrate a separate heuristic algorithm to quickly find a local optimum near the given possibly illegal solution, which is known as a memetic algorithm [82]. We propose a simple, but efficient heuristic for our purpose. While a massive set of modifications might lead to a good solution, it would be very costly. So we choose to make only obvious modifications which can be done in a short time and leave the large-scale exploration to the frame of outer probabilistic algorithm.

```

1 Fix_phase :
2 L = set of tiles with overflowing memory usage
3 while(L is not empty) {
4     T = tile with max overflow
5     M = set of MP communications to/from T
6     while(T has overflow) {
7         c = member of M with min gain
8         c.type = SM
9         decrease mem usage of c.src and c.dest
10        remove c from M
11    }
12    remove T and other non-overflow tiles from L
13 }
14 Optimize_phase :
15 L = set of all tiles
16 while(L is not empty and there are SM communications) {
17     T = tile with min remaining memory
18     M = set of SM communications to/from T
19     while(M is not empty) {
20         c = member of M with max gain
21         if(moving c to MP doesn't cause mem overflow in c.src
22         or c.dest) {
23             c.type = MP
24             increase mem usage of c.src and c.dest
25             remove c from M
26         }
27     }
28     remove T from L
29 }

```

Figure 2.8: Heuristic algorithm for communication type optimization.

In our model, it is probably better to utilize the space of local memories as much as possible for message passing unless the routing paths are very congested. Therefore, the problem looks similar to 0-1 knapsack problem, where a local memory corresponds to a knapsack and communication data correspond to the items to be put into the knapsack. The difference is that each item (communication data) does not occupy space in a single local memory, but occupies space in two memories (source and destination).

The idea of our heuristic is very simple; it tries to fill up the memories with

messages that give the most gain. The gain means increase of objective function values when the communication is mapped to MP (message passing). We define the gain as  $(\text{Hop\_distance}(\text{SM}) - \text{Hop\_distance}(\text{MP}))/m_c$ , where the numerator contributes to network energy consumption and latency. The pseudo-code of the proposed heuristic is given in Figure 2.8.

The algorithm is greedy and runs in two phases. The first phase is Fix phase. It starts with the initial mapping of tasks and communications given by the previous mapping stage. The initial mapping may have some tiles that need more local memory space than it has (memory overflow). Such an illegal mapping can be obtained because probabilistic algorithms like QEA may not consider constraints such as memory limitation when generating a solution. The Fix phase legalizes the initial mapping by visiting each tile with memory overflow and switching some message passing communications to shared memory. In this process, the order of tiles for visiting is important, because changing one communication type affects the tile on the other side. Intuitively, we choose to fix the tile with the most overflow first, because we do not want to modify the given solution unnecessarily much, and tiles with small overflows are likely to be resolved while fixing other tiles.

The second phase (Optimize phase) tries to fill up the remaining local memory space. In this phase, we visit tiles in the ascending order of remaining memory size. This means we handle tiles with smaller freedom first. If we handle tiles in the other order, memories with small remaining space may be filled up by communications with lower gain when handling other tiles.

Although the heuristic algorithm contains two doubly-nested loops, the time complexity is not as high as it seems. The Fix phase in the algorithm shown in Figure 2.8 contains a doubly nested loop with max/min operations. Inside the outer loop, the max operation in line 4 takes  $O(|P|)$  time because the size of

$L$  is upper-bounded by  $|P|$ . Since the outer loop is executed  $O(|P|)$  times, the overall time complexity of executing line 4 is  $O(|P|^2)$ . Inside the inner loop, the min operation in line 7 takes  $O(|C_p|)$  time, where  $p \in P$  is an index of each tile and  $C_p$  is the set of communications to and from tile  $p$ . Because the inner loop is executed at most  $|C_p|$  times and the min operation in line 7 takes  $O(|C_p|)$ , the time complexity of executing line 7 within an iteration of the outer loop is given by  $O(|C_p|^2)$ . Since the process is repeated for each tile (with overflowing memory usage) over the execution of the outer loop, the total time complexity of executing line 7 is given by

$$O\left(\sum_p |C_p|^2\right) \leq O\left(\sum_p |C_p|\right)^2 = O(|C|^2), \quad (2.8)$$

Therefore, the time complexity of the entire Fix phase is  $O(|P|^2 + |C|^2)$ . Because the structure of the Optimize phase is the same as the Fix phase and they run sequentially,  $O(|P|^2 + |C|^2)$  is the time complexity for the whole heuristic.

### 2.7.3 Design Space Pruning via Pre-evaluation

Once an initial mapping is made and further optimized with the heuristic algorithm, we schedule the tasks and packets to determine application throughput attainable with the current mapping. However, scheduling is a time-consuming process and it is the most significant bottleneck of our algorithm. Thus, before tackling the scheduling problem, we evaluate energy consumption and throughput lower bound. If the result does not satisfy the constraint on energy consumption or throughput, we prune the current mapping to avoid unnecessary scheduling. The conditions for pruning are as follows:

1. In the case that minimum energy consumption is given as a constraint, if the energy consumption calculated for the current mapping is larger than

the constraint, the current mapping is pruned.

2. In the case that minimum  $II$  is given as a constraint, if  $MII$  (Minimum Initiation Interval - a lower bound for  $II$ ) is larger than the constraint, the current mapping is pruned.
3. If the lower bound of objective function is larger than the best obtained so far, the current mapping is pruned.

In condition 1, the energy consumption for the mapping is calculated without scheduling by equation (2.1). In condition 2,  $MII$  is calculated by taking the larger of resource-constrained  $MII$  ( $resMII$ ) and recurrence-constrained  $MII$  ( $recMII$ ) which come from iterative modulo scheduling [83]. To obtain tight  $MII$ , we need to have a lower bound for network packet latency. For this, we use equation (2.4), which is a packet latency expected when the network is empty. Even if the constraints are satisfied, if the current objective function value is worse than the best obtained so far, the current mapping is pruned as in condition 3.

#### 2.7.4 Scheduling

A more accurate evaluation of a solution requires scheduling that determines the initiation interval. In this work, unlike the previous approaches, we apply software pipelining even in the presence of backward dependencies. For this, we adopt the iterative modulo scheduling [83], which is one of the most popular techniques for software pipelining. Starting from the  $MII$  (Minimum Initiation Interval), the iterative modulo scheduling performs height-based (priority-based) list scheduling where scheduling one task can possibly unschedule others. If it fails to find a schedule (with pre-determined amount of effort), the scheduler gives up  $II$  and goes on for  $II + 1$ .

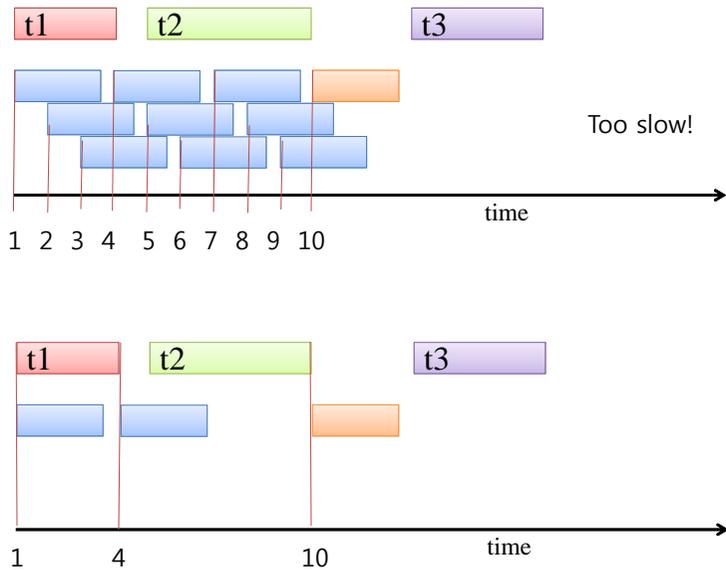


Figure 2.9: Concept of coarse-grained modulo scheduling.

Although iterative modulo scheduling<sup>5</sup> is proven to be efficient with empirical computational complexity of  $O(|T|^2)$ , applying it directly to our problem is impractical because it takes too much time for scheduling our task graphs with large number of cycles per task. Since we are dealing with very long execution times, trying to reschedule a task by increasing the schedule time one cycle by one cycle or trying to obtain a feasible solution by increasing  $II$  one by one will take huge amount of time. Considering that we are running the scheduling algorithm inside a design space exploration loop, direct application of iterative modulo scheduling is prohibitive. Thus we modify the scheduling algorithm to make it faster at the cost of loose schedule as explained in the next paragraph.

Figure 2.9 shows the concept of the modified iterative modulo scheduling, which we call coarse-grained iterative modulo scheduling, and Figure 2.10

<sup>5</sup>Original iterative modulo scheduling has been devised for scheduling operations instead of tasks and thus  $|T|$  represents number of operations.

```

1  Procedure IterativeSchedule(int II)
2  {
3    schedule start task at time 0;
4    Budget := Budget - 1;
5    //Budget is initially set to twice the # of tasks. If scheduler tries scheduling
      on current II more than budget times, try next II
6    while (unscheduled list is not empty) and (Budget > 0)
7    {
8      tsk := highest priority task in unscheduled list;
9      TimeSlot := FindTimeSlot(tsk);
10     schedule tsk at TimeSlot;
11     unschedule all operations conflicting with tsk;
12     Budget := Budget - 1;
13   }
14   if (unscheduled list is empty) return;
15   else IterativeSchedule(II + MinimumOverlap);
16 }

17 Procedure FindTimeSlot(task)
18 {
19   SchedTime = NULL;
20   CurTime := MinTime;
21   //find a time slot which yields no resource conflict
22   calculate MinTime and MaxTime by the task's predecessors;
23   while(CurTime < MaxTime && SchedTime = NULL)
24   {
25     if (resource conflict at CurTime)
26       CurTime := CurTime + exec. time of conflicting task;
27     //originally, CurTime++;
28     else SchedTime := CurTime;
29   }

30   //if finding a legal slot fails,
31   //schedule it at MinTime again or at next candidate
32   if(SchedTime = NULL)
33   {
34     if(task has never been scheduled or previously scheduled before MinTime)
35       SchedTime := MinTime;
36     else
37       SchedTime := next end time of another dependency later than
          PreviousSchedTime;
38     //originally, SchedTime := PreviousSchedTime + 1;
39   }
40   return SchedTime;
41 }

```

Figure 2.10: Pseudo-code of coarse-grained modulo scheduling.

shows the pseudo-code. It has two modifications to the original iterative modulo scheduling algorithm. The first modification is to speed up the process of finding a legal time slot where a task is to be scheduled. In the original iterative modulo scheduling, upon conflict with another task (conflicting task) already in the schedule, a legal time slot is found by linear-scanning of all time slots in the available window (time span between earliest bound and latest bound of possible time slots) cycle by cycle until a time slot without any conflict is found. Our modified algorithm eliminates the overhead of unnecessary conflict-checks by jumping to the 'end of conflict' for the schedule of the task instead of blindly checking every cycle. The 'end of conflict' means the end-time of the conflicting task on the same PE. If the conflict is due to the dependency from tasks on other PEs, it is right after the incoming packets from the predecessor tasks have arrived. For example, assume that the scheduler tries to schedule a new task A to a core where three tasks are already scheduled as in Figure 2.9. In the original scheduling algorithm, total ten timeslots have to be tested until finding an empty slot, whereas in the coarse-grained approach, only three slots have to be checked.

The same idea also applies to finding a time slot for the re-schedule of an already scheduled task. When there is no available time slot for a task in modulo scheduling, the task is scheduled at the earliest possible time slot (`minTime`) disregarding any resource conflict and unschedules the conflicting tasks (Figure 2.10 line 34-35 and line 11). Thus the unscheduled task should be re-scheduled. When it is re-scheduled, the scheduler looks for a legal time slot just like normal scheduling. If a legal time slot is not found for the re-scheduling task, then it is again scheduled at the `minTime` and the conflicting ones are unscheduled. Here, possibility of livelock comes from unscheduled tasks being scheduled in the same time slot over and over again. To avoid this, the

scheduler has to keep track of the “previously scheduled time” for each task. If a task is about to be re-scheduled at “previously scheduled time” again, it is then alternatively scheduled to a different slot. In original modulo scheduling, the alternative slot is “previously scheduled time + 1” (Figure 2.10 line 38). Although this might find a schedule closer to the optimal solution, we have found it too slow for our purpose. Instead, our algorithm tries the earliest one, among the end times of tasks that cause resource conflict and arrival times of the packets which the task has dependency on (Figure 2.10 line 37).

As for the second modification, when the algorithm has to give up the current  $II$  and increase it, instead of adding “1”, it adds the minimum conflict overlap that has occurred while trying to schedule for the current  $II$ . Again, this scheme might not find optimal  $II$ , but increasing  $II$  one by one leads to unrealistically long running time, especially when scheduling tasks of long execution time. Our idea is based on the observation that, in many cases where two tasks have a conflict with each other during scheduling, the conflict is resolved by increasing  $II$  by the amount of conflict. So we try adding the minimum amount of conflict occurred during scheduling to the current  $II$  (Figure 2.10 line 15).

Together, these modifications significantly reduce the running time of the scheduler at the cost of slightly loose (but still valid) schedule. The effect of the coarse-grained iterative modulo scheduling will be discussed in Section § 2.8.1.

## **Packet Scheduling**

For an accurate timing model, we choose a deterministic packet scheduling approach. The scheduler is integrated into the coarse-grained modulo scheduling framework. Basically, packets can be scheduled in the same way as tasks with a simple extension. The process of transferring a packet through a single hop

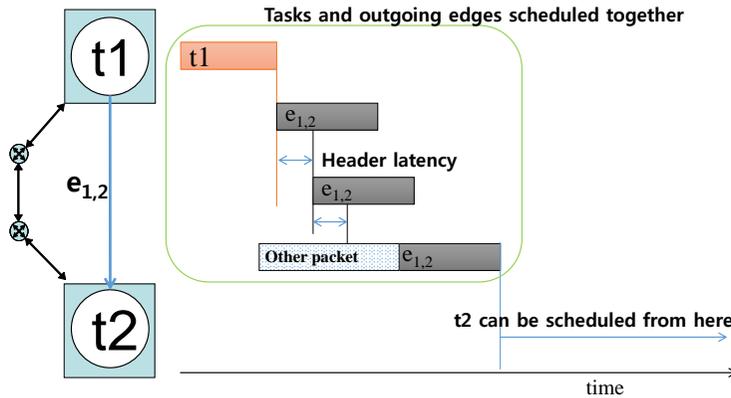


Figure 2.11: Diagram showing packet scheduling.

link, which we call link transfer, can be considered as an individual task that has an incoming dependency from the link transfer of the previous hop and an outgoing dependency to the link transfer of the next hop. Of course, the first link transfer has dependency from the packet's source task and the destination task has dependency from last link transfer of the packet. Then each link transfer occupies a link during the transfer of the packet, just like a task occupying a PE during its execution.

However, scheduling packets in the same way as task scheduling may slow down the scheduling significantly for two reasons. First, scheduling a packet transfer from the source to the destination generally involves scheduling too many link transfers (the total number of link transfers will be the number of packets times average hop count). Considering that there are usually many hops in a communication transaction, this can be a serious problem that slows down the scheduling algorithm. Secondly, modulo scheduling assumes that tasks can wait arbitrarily long until all the dependencies and conflicts are resolved. If we do the same for packet scheduling, packets might have to stay at routers for a long time. This logically does not make a problem, but practically, the

buffer space required at the routers increases too much. Furthermore, we have found that allowing packets to stay in the network for an arbitrarily long time complicates the schedule and will be an extra burden to the scheduler. To avoid these problems, we set the following rules for scheduling packets.

1. When a task is scheduled, all outgoing packets from that task are scheduled together from the source to each destination. Similarly, when a task is unscheduled, all the outgoing packets from the task are unscheduled together.
2. Unlike tasks, packets cannot unschedule individual link transfers of other packets. Thus, when the attempt at scheduling a packet fails, it is regarded as failed instead of unscheduling conflicting link transfers.
3. For the scheduling of link transfers, the scheduler keeps track of the buffer space usage at each port. A link transfer cannot be scheduled if scheduling at some time slot would result in buffer overflow.

Rule 1 prevents the scheduler from calculating the priority and dependencies of each link transfer. Rule 2 ties all link transfers of a packet together as a single instance while maintaining characteristics of wormhole routing. Without this rule, when a packet fails to find a time slot, it will try to find one by unscheduling an already scheduled link transfer. Then the unscheduled link transfer will be scheduled at a later time slot, possibly unscheduling all other link transfers of the same packet and forcing them to be scheduled again, incurring significant overhead in the scheduler runtime. These rules effectively make the packets to be seen as ‘tails’ of the tasks. Although the packet scheduling looks quite sloppy compared to the task scheduling, it does not degrade the quality of the obtained schedule much unless communication time dominates computation time, which

is not likely in many cases. Rule 3 is to resolve the second problem stated above. Figure 2.11 shows a simple example of packet scheduling. Assume  $t1$  has an outgoing edge (communication)  $e_{1,2}$  to  $t2$ . When  $t1$  gets scheduled,  $e_{1,2}$  gets scheduled together. First,  $e_{1,2}$  is scheduled on the link between PE and the first router. After the delay of the header propagation, it can be scheduled on the next link. Of course, if the time slot is already occupied by other packet(s), the packet for  $e_{1,2}$  becomes blocked and scheduled after that. Task  $t2$  can be scheduled only after the arrival of  $e_{1,2}$ .

## 2.8 Experimental Results

In this section, we show a series of experimental results. First, we show effectiveness of our algorithm design choices. In § 2.8.1, we show comparison of our scheduling method with existing speedup techniques. In § 2.8.2, our mapping algorithm is compared with other existing ones to justify that our algorithm is competitive even disregarding communication type mapping. The overall result is shown in § 2.8.3. § 2.8.4 and § 2.8.5 are dedicated to sensitivity analysis to see how local memory size and placement of shared memory(s) can affect the mapping solutions.

### 2.8.1 Experiments with Coarse-grained Iterative Modulo Scheduling

To justify using the coarse-grained iterative modulo scheduling that we explained in § 2.7.4, we compared our approach with a variation of the original

Table 2.2: Comparison of Scheduling Methods

	Original+ [84]	Proposed	Ratio
Average time spent on scheduling (sec)	2.25	0.0043	1/520
Average II obtained (cyc.)	61998.6	62921.0	1.015

iterative modulo scheduling, which increases  $II$  by 1% of  $MII$  for speedup [84]. The scheduling was done with packet scheduling as well as task scheduling. We ran the scheduling algorithm on 1,000 arbitrary inputs for both original algorithm and coarse grained algorithm. The result is shown in Table 2.2.

As can be seen in the table, the coarse-grained scheduling is an order of magnitude (more than 500 times) faster than the simple speedup technique at the cost of 1.5% degraded scheduling result. In fact, the original scheduling takes too much time to be used in our meta-heuristic algorithm that performs at least a few hundred thousands of schedulings. Of course, this result does not always apply in the exactly same ratio because the speedup depends on application size and other characteristics.

## 2.8.2 Comparison with Different Mapping Algorithms

Although our main contribution is not in the mapping algorithm itself, we need to show that the mapping algorithm we used generates a mapping solution that is at least comparable in its quality to previous ones. For this purpose, we have compared QEA based mapping algorithm with three other mapping algorithms: Random mapping, NMAP [51], and SA (Simulated Annealing). NMAP first builds up an initial solution using the LCF (Largest Communication First) scheme, which picks the task that has the largest communication volume with tasks that are already mapped. The chosen task is then placed at the tile that gives the best partial result. After that, NMAP gives each task one chance of swapping with another task for further improvement. SA is one of the most popular algorithms for mapping [54,55]. For SA, we also used LCF as an initial mapping and a simple composition of movements similar to [85] is used. The movements are as follows:

- M1: Move a task to another PE.

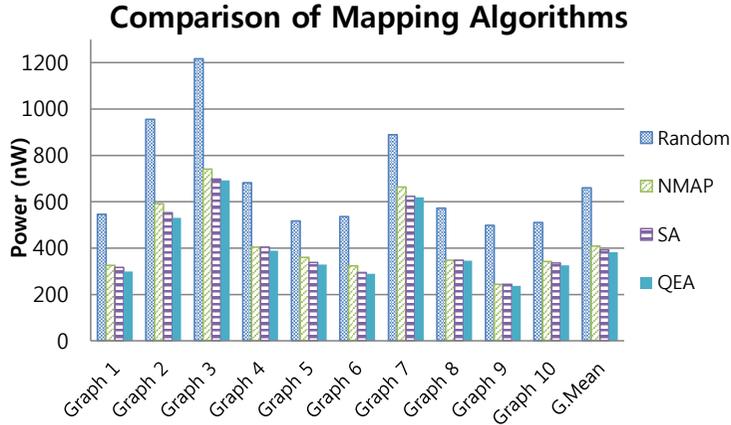


Figure 2.12: Experimental results on different mapping algorithms.

- M2: Swap two task's locations.

where M1:M2 was empirically set to 5:2. Also, linear cooling schedule was used and  $\exp((P - P')/T_{mp})$  was used as the acceptance probability function where  $P, P'$  and  $T_{mp}$  correspond to old solution quality, new solution quality, and current temperature. A windowing scheme was applied as in [85] to allow large scale movements at earlier stages and only small changes near the end.

Because all the algorithms for comparisons were designed for core mapping which targets low power consumption, our algorithm was also run for core mapping only and scheduling was omitted. Also, for fair comparison, the parameters of QEA and SA were set so that their runtimes were about the same. All mapping algorithms assumed XY routing, and 10 random core graphs obtained from TGFF [81] was used for an input set. Figure 2.12 shows the results. Clearly, our QEA based algorithm performs best among the algorithms. In geometric mean, solution from our algorithm is about 6.8% better than NMAP and 2.7% better than SA. This shows that our algorithm finds relatively good mappings. The difference is not significant, but we do not have any reason not to use

the algorithm which shows the best result. Runtimes of QEA and SA were less than a minute, whereas that of NMAP was a few seconds. Even though NMAP is relatively fast, such a heuristic is hardly modified to support additional dimensions including communication types. Furthermore, searching the resulting larger solution space with such a heuristic would not guarantee any better performance.

### 2.8.3 Experiments with Overall Algorithms

To show the effectiveness of our approach, we chose a set of applications to be tested. Table 2.3 shows their statistics. Two applications have cycles (backward edges) in them and others are acyclic. Applications were converted to task graphs to exploit task parallelism. Mapping and scheduling were done for each of the three objectives: energy consumption,  $II$ , and EDP. The running times were tens of minutes to a few hours depending on the size of the application graphs, when we ran our mapping and scheduling algorithm on a computer with an Intel 2.67 GHz i7 processor. The size of each local memory was set to 4KB, which was not large enough for the applications to have every communication through message passing.

The results are shown in Figure 2.13. The bars represent the results of the algorithm for reducing (a) energy consumption, (b)  $II$ , and (c) EDP. In the graph, “SM only” bars represent the results obtained from the mapping and schedul-

Table 2.3: Test Set Applications

	Histogram	Matrix Multiply	Mergesort	MPEG4 Decoder	MJPEG Decoder	Synth1	Synth2
Nodes	33	112	63	11	7	31	20
Edges	32	192	62	14	6	47	40
Backward Edge	No	No	No	Yes	No	Yes	No

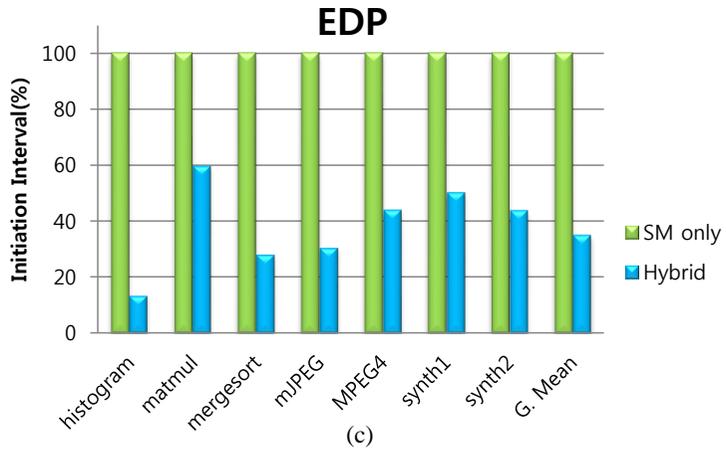
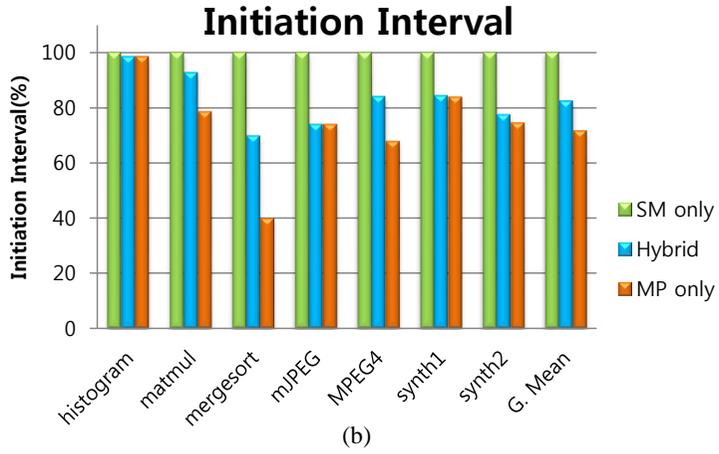
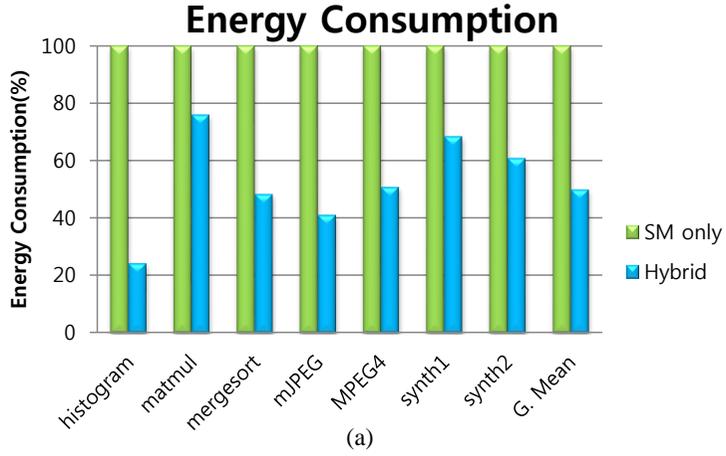


Figure 2.13: Mapping and scheduling results. (a) for energy consumption, (b) for II and (c) for EDP.

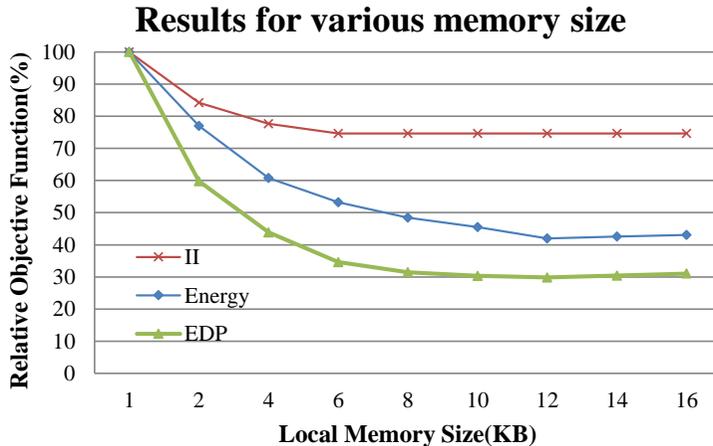


Figure 2.14: Experimental result for various local memory sizes.

ing with all communications using shared memory and no communication type optimization. “Hybrid” bars represent results obtained by our approach. Figure 2.13 (b) also shows “MP only” bars for the case where all communications take message passing. In this case, local memory constraints are ignored. This result shows the lower bound of  $II$  that could be obtained for each application. Since the performance and energy consumptions of the applications vary greatly, we normalized the results to the SM case. When the algorithm targets energy reduction, we could obtain 50.1% lower energy consumption in geometric mean by the hybrid approach. When targeting  $II$  reduction, we obtained 21.0% higher application throughput ( $1/II$ ) by hybrid approach in geometric mean. For energy-delay product, the hybrid approach obtained 64.9% reduction in geometric mean.

#### 2.8.4 Experiments with Various Local Memory Sizes

Clearly, if the PE tiles have the larger local memory size, we have the more freedom for task allocation and communication type mapping, possibly obtaining

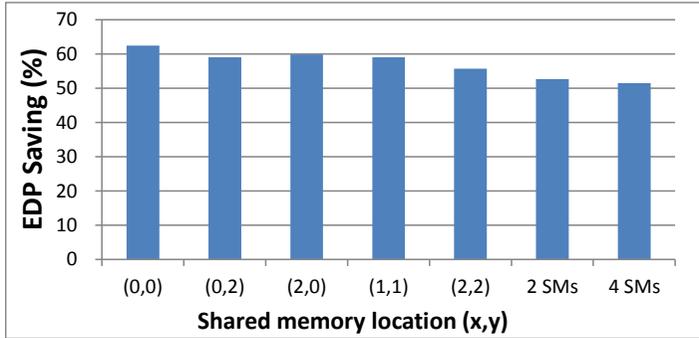


Figure 2.15: Experimental results on various/multiple shared memory locations.

better results for our objective functions. Figure 2.14 shows this experiment. Here, we used a random graph with 31 nodes and 47 edges, and varied the local memory size for each tile from 1KB to 16KB. No possible solution was obtained under 1KB because some tasks required larger memory space for their own data<sup>6</sup>. All three metrics ( $II$ , energy, and EDP) were improved as we increased the local memory size. However,  $II$  did not improve for local memory above 6KB because at this point most of the communications were done in message passing and no more improvement could come from communication type change. Moreover, both energy consumption and EDP start to increase rather than decrease when the local memory size exceeds 12KB because no better task/communication type mapping can be found above this point and larger memory consumes more energy.

### 2.8.5 Experiments with Various Placements of Shared Memory

Even though we assumed our architecture to have one global shared memory at the center, we could assume differently and place the global shared memory on other locations or place multiple shared memories. Figure 2.15 shows how the

<sup>6</sup>For simplicity, we assumed the local data of tasks to be on the local memories. Although it can be placed on the shared memory, we leave it as a future work.

placement of global shared memory affects the EDP savings of hybrid mapping and scheduling solutions of a random task graph compared to “SM only” solution. For the locations, we considered five different coordinates: at the corner (0,0), at the upper boundary (2,0), at the left boundary (0,2), at the center (2,2) and between center and the corner (1,1). For multiple shared memories, we used two shared memories at (1,1) and (3, 3), and four shared memories at (1,1), (3,1), (1,3) and (3,3).

Over the different configurations, the benefits are not much different, but we found that the savings are slightly lower in multiple shared memory architectures. This is because “SM only” solutions benefit more from multiple shared memories by shortened communication paths. Qualities of hybrid solutions using multiple shared memory blocks were worse than using single shared memories. The reason is that because of shared memory blocks, average message passing path length was increased and number of usable processors was decreased. Thus increasing number of shared memory blocks was not always helpful.

# Chapter 3

## Thermal Management

### 3.1 Introduction

In this chapter, we propose a DVFS technique for thermal management which controls both network routers as well as the cores. Even though there are many researches on using DVFS for thermal management, they are limited to core-only technique. However, it is known that networks also take a large portion of the total system power, and therefore has a significant impact on the temperature. By exploiting the thermal characteristics 3D ICs along with the application behaviors, we propose a runtime thermal management framework, *THOR* (named after *Thermal Orchestration*) to set proper operating points for each router and core which tries to obtain maximum performance under the thermal constraint.

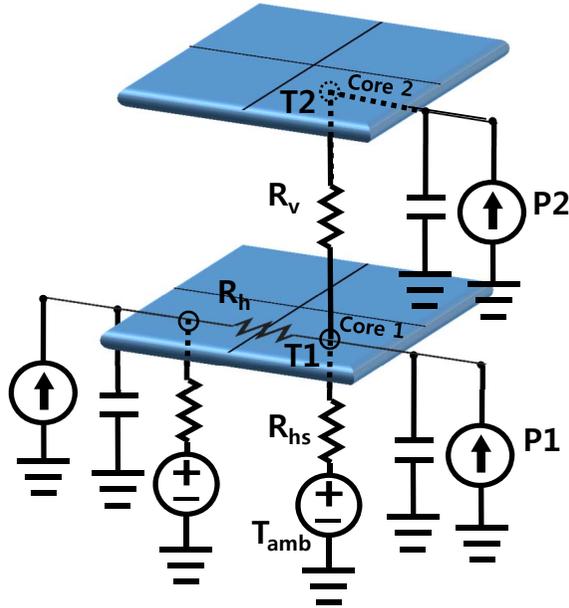


Figure 3.1: An example thermal RC model of a 3D CMP.

## 3.2 Background

### 3.2.1 Thermal Modeling

Thermal characteristics of a CMP and its package are usually modeled using an equivalent thermal RC model [86]. An example of a simplified thermal RC model is shown in Figure 3.1. In the thermal RC model, the CMP is logically divided into multiple blocks, and each block is represented by a junction (marked as circles in Figure 3.1) in the RC circuit. Temperature and heat flow respectively correspond to voltage and current in the electric circuit. Thermal resistance is proportional to the length of the path and inversely proportional to the cross-sectional area of the path. Thermal capacitance at a junction, on the other hand, is proportional to the product of thickness and area. Altogether, the circuit allows calculation of both steady-state and transient temperature of the system.

### 3.2.2 Heterogeneity in Thermal Propagation

There are two interesting aspects of thermal characteristics for 3D ICs: *thermal coupling* and *heterogeneous cooling efficiency*.

- 1) Thermal coupling: Dies used to build 3D ICs are usually thinned down to around hundred micrometers. Thus the vertically adjacent cores exhibit a very strong thermal correlation compared to laterally adjacent cores. In terms of thermal resistance, inter-layer resistance  $R_v$  is 0.13 K/W, whereas intra-layer resistance  $R_h$  is 6.67 K/W (more than 50 times higher) according to our data.
- 2) Heterogeneous cooling efficiency: Each die of stacked ICs shows different cooling efficiency due to its distance from the heatsink. Obviously, the die closest to the heatsink has the best cooling efficiency and the one farthest from the heatsink has the worst cooling efficiency. According to the simplified model of Figure 3.1, where tile 1 is closest to the heatsink and tile 2 is next to it, the steady-state temperature of tile 1 and tile 2 can be expressed as follows:

$$T_{ss}(1) = T_{amb} + (P_1 + P_2) \cdot R_{hs} \quad (3.1)$$

$$T_{ss}(2) = T_{amb} + (P_1 + P_2) \cdot R_{hs} + P_2 \cdot R_v \quad (3.2)$$

where  $T_{ss}(1)$  and  $T_{ss}(2)$  represent steady-state temperature of tile 1 and tile 2, respectively, and  $P_1$  and  $P_2$  correspond to power consumption of tile 1 and tile 2, respectively. As can be seen from the equations, steady-state temperature of the core far from the heatsink (core 2) is always higher than the one closer to the heatsink (core 1). Please note that we use in this work a further refined model for the experiments.

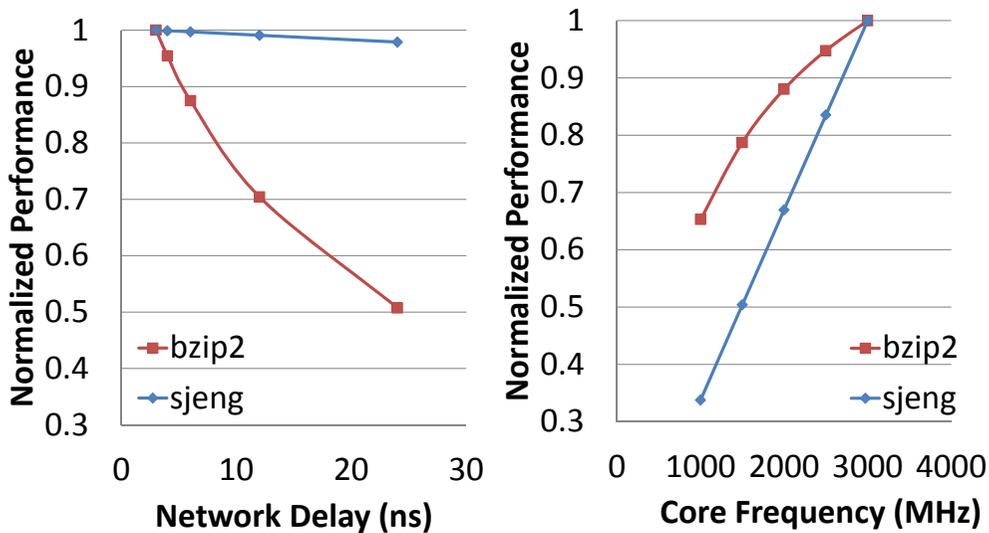


Figure 3.2: Normalized application performance numbers for different core/network speeds.

### 3.3 Motivation and Problem Definition

Network routers not only contribute to temperature increase, but also plays a major role in throttling the performance of the connected core [87]. However, the amount of the influence is different according to the characteristics of the application that runs on the core. For example, Figure 3.2 shows normalized performance numbers obtained by sweeping core/network speeds for two applications from the SPEC CPU2006 benchmark. In case of compute-intensive applications like sjeng, scaling down core frequencies has a large impact on the performance (as well as the power consumption and therefore temperature). However, slowing down the network does not affect the performance significantly. For memory-intensive applications such as bzip2, the behavior appears to be the opposite.

Most of existing DVFS approaches focusing on per-core DVFS move the

operation point only along the x-axis of the graph, while the approaches that adjust network consider moving it only through the y-axis. Our work, on the other hand, seeks for the optimal point on the whole xy-plane.

This work targets a 3D many-core tiled architecture connected via on-chip network with runtime support for per-core and per-router DVFS. We assume that each tile is equipped with a thermal sensor for temperature monitoring. Our approach assigns a v/f (voltage/frequency) pair to each core/router such that the performance of the system is maximized under a given thermal constraint. The notations are listed in Table 3.1. Formally, our problem can be described as determining  $f_c$  and  $f_r$  for each tile (the voltages are determined automatically once the frequencies are determined),

$$f_c(\text{core}) \in F_c \quad \forall \text{core} \in C \quad (3.3)$$

$$f_r(\text{router}) \in F_r \quad \forall \text{router} \in R \quad (3.4)$$

which maximizes the weighted speedup  $WS$  given by

$$WS = \sum_i \frac{IPS_i}{IPS_i^{ref}} = \sum_i SU_i \quad (3.5)$$

under the constraint of

$$T(\text{core}) < T_{max} \quad \forall \text{core} \in C \quad (3.6)$$

where  $IPS_i^{ref}$  represents performance (instructions per second) of application  $i$  when executed alone with reference frequency/voltage, and  $IPS_i$  represents performance in shared run with DVFS. We use instructions per second (IPS) instead of instructions per cycle (IPC), because clock period varies in systems with DVFS, leaving little meaning in IPC. Also, we use weighted speedup [88] as the performance metric instead of aggregate IPS to avoid favoring high-IPS

Table 3.1: Notations Used In the Paper

Symbol	Meaning
$C$	Set of all cores
$R$	Set of all routers
$L$	Number of dies stacked
$F_c$	Set of available operating frequencies of cores
$F_r$	Set of available operating frequencies of routers
$f_c(i)$	Operating frequency of core $i$
$f_r(i)$	Operating frequency of router $i$
$WS$	Weighted speedup
$SU_i$	Speedup of application $i$
$CPI_0$	Cycles per instruction measured with perfect L2 cache
$MPI_{L2}$	L2 cache misses per instruction
$D_{DRAM}$	Average DRAM access latency, including routing delays of the memory layer
$D_{net}$	Network round-trip latency
$h$	Per-hop latency of a network router in router cycles
$T_{ss}(i)$	Steady-state temperature of core $i$
$T(i)$	Measured temperature of core $i$
$T_{amb}$	Ambient temperature
$T_{max}$	Maximum allowed temperature
$R_v$	Inter-layer thermal resistance between cores
$R_h$	Intra-layer thermal resistance between cores
$R_{hs}$	Thermal resistance between the bottom-most core and the ambient (through the heatsink)
$P_i$	Power consumption of application $i$
$W_i$	Weight of power consumption for layer $i$
$WP$	Weighted-power (see § 3.5.2)
$u[n]$	PI controller output (=assigned weighted-power budget) of the control interval $n$
$e[n]$	Difference between reference and the measured temperature of the control interval $n$
$K_I$	Integral term constant of PI controller
$K_P$	Proportional term constant of PI controller

applications.

## 3.4 Related Work

Thermal problems of ICs have been taken seriously for decades, and there is a large pile of researches. The work done by Brooks and Martonosi [89] is one of the early literature that raised the problem of dynamic thermal management. The authors argued that cheaper packaging can be used by controlling CPU activity rather than designing for worst-case power consumption. Afterwards, many researches came out with various techniques. Donald and Martonosi [90] investigated various DTM techniques including DVFS and migration.

There is also a stream of work that considers 3D integration into account. Zhu et al. [91] proposed a thermal-aware operating system with thread migration and frequency assignment. Zhou et al. [92] suggested a task scheduling method for 3D architectures. Kang et al. [93] proposed a DTM considering both thermal and power constraint, with ability to exploit temperature slack. Some work tries to solve thermal problems in NoC. Shang et al. [28] showed the significance of thermal problem in network with its modeling, and Chao et al. [94] proposed a thermal management in 3D NoC with routing protocol and throttling method. Also, [95] suggested controlling last-level caches and NoC together within a single v/f domain to optimize energy consumption.

## 3.5 Orchestrated Voltage-Frequency Assignment

### 3.5.1 Individual PI Control Method

As a first step of v/f assignment, we can think of each core and router performing frequency/voltage control individually according to the sensed temperature. We employed the PI control method as in many state-of-the-art systems deploying DVFS [90, 95–97]. Basically, a PI controller sets a target variable such as the

clock frequency of a core every pre-determined interval based on a measured variable such as the temperature as well as the history of the variable. The most beneficial advantage of a PI controller is that it can adapt to a certain degree of unexpected changes such as modeling errors or runtime temporal variation of applications, especially compared to systems with rule-based approaches which rely on exact modeling [28, 91].

The discrete form of the PI controller is given by

$$u[n] = u[n - 1] + K_I \cdot e[n] + K_P \cdot (e[n] - e[n - 1]) \quad (3.7)$$

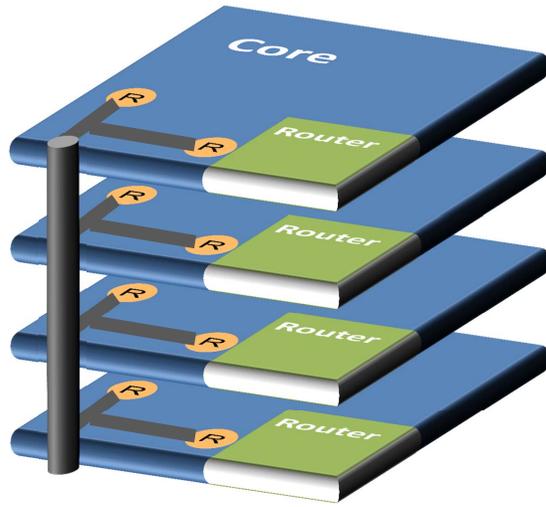
where  $e[n]$  is the difference between  $T_{max}$  and  $T$  in the  $n^{th}$  control interval. In other words,

$$e[n] = T_{max} - T[n] \quad (3.8)$$

and  $u[n]$  is the output of the PI controller. In the individual PI control method, each core and router has its own PI controller and  $u[n]$  is the frequency ratio with respect to its full speed. At the end of each control interval, all cores and routers calculate their  $u[n]$  values using the temperature from the thermal sensors to set their operating speeds and run for the next interval.

### 3.5.2 PI Controlled Weighted-Power Budgeting

While the individual PI control method can acquire the objective of thermal control using both core and network DVFS, it does not consider thermal correlations between tiles. More precise and efficient control over the components would be possible if we consider multiple tiles together. However, we cannot consider all tiles at once since it would be too costly when there are many cores (or tiles) in the system. Thus we perform grouping of tiles and consider the thermal correlation between tiles only within each group. A natural way is to group the tiles in a pillar, which is a set of tiles on the same horizontal



**Power supply**

Figure 3.3: The internal structure of a pillar.

position. As described in Subsection § 3.2.2, the thermal correlation between vertically adjacent cores is much stronger than that between laterally adjacent cores. Also, the approach is scalable because the number of layers stacked does not grow much (typically several layers). Thus treating tiles in a pillar independently from the tiles in other pillars can effectively reduce the complexity of the problem while retaining a proper level of accuracy [91–93]. Figure 3.3 shows the internal structure of a pillar. Each pillar has a few stacked layers (four in this work), and each layer has a core and a router. Each core and router has a voltage regulator to perform DVFS individually.

Then what is the target quantity set by  $u[n]$  under grouping? In the easiest way, it could be the target frequency of all the cores/routers within the pillar. However, since all the components in the pillar have the ability to be controlled differently, having a single frequency level of all the components as the controller output would be a complete non-sense. In this work, we propose weighted-

power budget to be used for the PI-controller output of pillars. Since we are considering the maximum temperature as the constraint, it is good enough to focus on the top core of the pillar, the hottest one. Neglecting the minimal heat flowing through lateral paths, the steady-state temperature of the top core is characterized by equation ((3.2)). Re-writing the equation yields

$$T_{ss}(2) = T_{amb} + (R_v + R_{hs}) \cdot P_2 + R_{hs} \cdot P_1 \quad (3.9)$$

which shows that power consumption of each tile contributes differently to the temperature. Thus, budgeting power for each pillar by mere aggregation is not adequate. Instead, we perform budgeting by weighting power consumption in a tile with thermal resistance from that tile to the heatsink. Then the weighted-power  $WP$  is defined as follows:

$$W_l = (l - 1) \cdot R_v + R_{hs} \quad (3.10)$$

$$WP = \sum_{l=1}^L (W_l \cdot P_l) \quad (3.11)$$

By budgeting the weighted-power in this manner, we can set the target steady-state temperature for the next control interval. We take advantage of PI control to adaptively determine the weighted-power budget according to current system status (i.e., controller output  $u[n]$  is the budget of  $WP$ ).

### 3.5.3 Performance/Power Estimation

Once the weighted-power budget has been determined for each pillar, operating points (v/f pairs) of the cores and routers have to be decided such that the weighted power is kept within the budget. Thus the problem of frequency assignment defined in Section § 3.3 is reduced to frequency assignment in each pillar, where the number of tiles is now reduced to the number of layers and weighted-power constraint is used instead of thermal constraint. To determine

proper operating points, we need the speedups and power consumptions that will result from different settings of operating points. Unfortunately, since such prior knowledge is not available, we rely on an estimation model derived based on hardware statistics monitored at runtime.

For the derivation of the model, we assume that each core has private L1/L2 caches and is connected to the memory controller on the top layer via NoC. When accessing data in the DRAM, we chose a routing algorithm that routes the packets along the path within the pillar and uses xy routing on the top layer. In this way, the application speedup can be isolated from the speed of the routers in other pillars. To calculate the speedup, CPI of an application can be expressed as follows:

$$CPI(core) = CPI_0 + MPI_{L2} \cdot Latency_{memory} \quad (3.12)$$

where  $CPI_0$  represents the cycles per instruction measured with perfect L2 cache and  $MPI_{L2}$  represents L2 cache misses per instruction. When average DRAM access delay and network delay are known,

$$CPI(core) = CPI_0 + MPI_{L2} \cdot (D_{DRAM} + D_{net}) \cdot f_c(core) \quad (3.13)$$

Since  $IPS = \frac{f_c}{CPI}$ , the speedup can be calculated as follows:

$$\begin{aligned} SU(core) &= \frac{IPS}{IPS_{ref}} \\ &= \frac{f_c(core)}{f_{c.ref}} \cdot \frac{CPI_0 + MPI_{L2.ref} \cdot (D_{DRAM} + D_{net.ref}) \cdot f_{c.ref}}{CPI_0 + MPI_{L2} \cdot (D_{DRAM} + D_{net}) \cdot f_c(core)} \end{aligned}$$

To calculate  $D_{net}$ , we take the sum of hop latencies for the round-trip from the core to the memory controller layer. The one-way network delay for a memory access is measured from the injection time of a packet to the time when the packet arrives at the top layer along the path including  $n$  routers and  $n$  links

when the source node is  $n$  hops away from the top layer. Thus when the core is located at layer  $l$ ,

$$D_{net}(l) = \sum_{i=0}^l \frac{2 \cdot h}{f_r(i)} \quad (3.14)$$

where  $h$  represents the hop latency of a router in cycles. This model does not take the congestion delay into account. However, NoCs are usually designed to have low to medium load. We have found that in our experiments, congestion delay does not play a significant role.

The obtained estimation of  $SU$  can be used to estimate the power consumption of the components, which is proportional to the switching activity. For cores, the switching activity is proportional to the speedup. For routers, it is proportional to the number of flits routed. Because the control interval is relatively long, the number of packets averaged over an interval does not differ significantly from that of the next interval (burst traffic is hidden by the averaging effect). In the long run, however, the statistics may change significantly due to various reasons such as thread migration or other context switching. Fortunately, the control interval is short enough (see § 3.6.1) to avoid the problem. Thus we assume that the number of flits passing through the router is similar to that of the last interval, which has been observed in our simulation most of the time. From these, we can estimate the dynamic power consumptions of the cores and routers. Also, leakage power of both cores and routers is proportional

to voltage [98]. Therefore, the estimations of the power consumptions are:

$$dyP_{core} = dyP_{core.ref} \cdot SU \cdot \left( \frac{v_{core}}{v_{core.ref}} \right)^2 \quad (3.15)$$

$$dyP_{router}[n] = dyP_{router}[n-1] \cdot \left( \frac{v_{router}[n]}{v_{router}[n-1]} \right)^2 \quad (3.16)$$

$$stP_{core} = stP_{core.ref} \cdot \frac{v_{core}}{v_{core.ref}} \quad (3.17)$$

$$stP_{router} = stP_{router.ref} \cdot \frac{v_{router}}{v_{router.ref}} \quad (3.18)$$

and finally,

$$P = dyP_{core} + dyP_{router} + stP_{core} + stP_{router} \quad (3.19)$$

where  $P$  is the total power consumption of the tile. Although not perfectly accurate, the proposed model can quickly estimate power and performance with acceptable accuracy. We show the accuracy of the model in § 3.6.3.

### 3.5.4 Frequency Assignment

Utilizing the estimation model for performance and power, the frequency/voltage pairs for all the components in the pillar have to be decided. Possible number of combinations for the problem is  $(|F_c| \cdot |F_r|)^L$ . In case of five operating levels for each of the cores/routers in four layers, the number becomes  $25^4=390625$ . This makes exhaustive search inappropriate for on-line control decision.

In this work, we adopt a hill climbing approach to rapidly solve the problem. As shown in Figure 3.4, the approach starts from the operating points used during the last interval. If the current weighted-power is higher than the budget, then, for each component, the performance/power is estimated with its operating point lowered by one step. Among the adjustments, the one that gives the minimum cost (cost=(speed loss)/(power reduction)) is chosen. The pro-

```

1 Procedure freq_assign (budget, pillar)
2 (WS, WP) = estimation_model(Fc, Fr)
3 while WP >= budget
4   for i in pillar.components
5     (Fc', Fr') = lower_one_step(i, Fc, Fr)
6     (WS', WP') = estimation_model(Fc', Fr')
7     cost = (WS-WS')/(WP-WP')
8     if cost < mincost
9       mincost = cost
10      minF = (Fc', Fr')
11      minSP = (WS', WP')
12    end if
13  end for
14  (Fc, Fr) = minF
15  (WS, WP) = minSP
16 end while
17 while true
18   for i in pillar.components
19     (Fc', Fr') = higher_one_step(i, Fc, Fr)
20     (WS', WP') = estimation_model(Fc', Fr')
21     gain = (WS'-WS)/(WP'-WP)
22     if gain > maxgain && WP' < budget
23       maxgain = gain
24       maxF = (Fc', Fr')
25       maxSP = (WS', WP')
26     end if
27   end for
28   if (Fc, Fr) == maxF
29     break
30   else
31     (Fc, Fr) = maxF
32     (WS, WP) = maxSP
33   end if
34 end while
35 return maxF

```

Figure 3.4: Hill-climbing algorithm for frequency assignment.

cess is repeated from that adjusted combination until the weighted-power goes below the budget. Then similar process is repeated to set the operating speed higher, taking the combination with the most gain (gain=(speed gain)/(power increase)) while not exceeding the weighted-power budget. After the final combination is decided, the frequency/voltage pairs of the cores/routers are actually adjusted for the current control interval.

This heuristic works well since the speedup and power consumption are monotone increasing functions of core and router frequencies. We found that the quality of this method is close to that of the exhaustive search. While the theoretical upper bound of the computational complexity<sup>1</sup> is  $O((|F_c| + |F_r| - 2) \cdot 2L^2)$ , the practical runtime is much lower because the operating speed seldom needs a drastic change. Note that each decision is local to a pillar and can be made in parallel.

### 3.5.5 Algorithm Overview

Figure 3.5 shows the overall structure of the algorithm, which is run for each pillar independently. First, the PI controller uses the sensed temperatures of the components to determine weighted-power budget for this interval. Taking the budget and monitored hardware statistics ( $MPI_{L2}$  and  $D_{DRAM}$ ) from the system, operating points for the components are determined using the estimation model. Then, the operating points of the components in the pillar are set as decided, and run for the current interval. The statistics and temperature are monitored during the current interval and fed back to calculate operating points for the next interval.

---

<sup>1</sup>The maximum number of steps required to move all ( $L$ ) frequency pairs from frequency levels  $(0, 0)$  to  $(|F_c| - 1, |F_r| - 1)$  is  $(|F_c| + |F_r| - 2) \cdot L$ . Each step, we have to check  $2L$  combinations ( $L$  for cores,  $L$  for routers) and this results in  $((|F_c| + |F_r| - 2) \cdot 2L^2)$ .

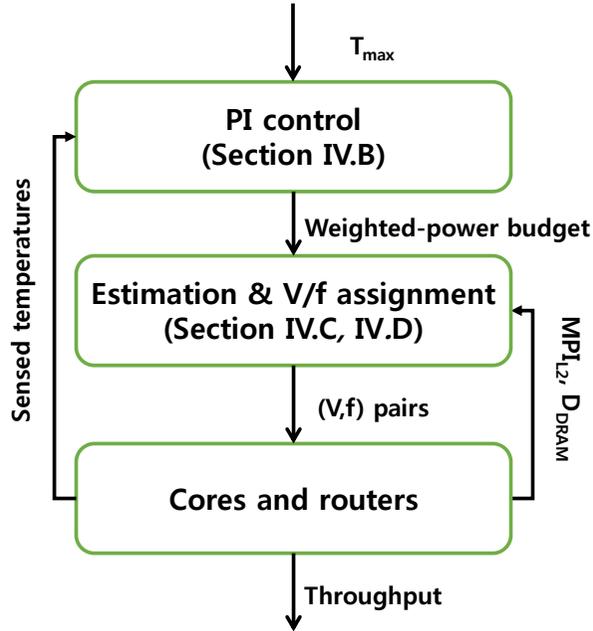


Figure 3.5: Overall structure of the algorithm.

### 3.5.6 Stability Conditions for PI Controller

One important issue in designing a PI controller is ensuring stability of the system. The system is stable if and only if all poles of the transfer function lie within the unit circle of the  $z$ -plane. Applying Routh-Hurwitz criterion to equation ((3.7)) and ((3.8)), the conditions that ensure stability becomes (we omit the detailed derivation here)

$$1) \quad K_I + 2K_P < 2 \cdot \frac{1 + \rho}{1 - \rho} \quad (3.20)$$

$$2) \quad K_P > -1 \quad (3.21)$$

$$3) \quad K_I > 0 \quad (3.22)$$

where  $\rho$  is the time constant related to the RC value of the circuit ( $= e^{-\Delta t/RC}$ ). The three conditions together form a triangle in  $K_I$ - $K_P$  plane, and we can apply

the standard pole placement method [99] to determine the parameter values.

## 3.6 Experimental Result

### 3.6.1 Experimental Setup

Table 3.2 shows system design parameters that we used for evaluation. We used Sniper [100] multi-core simulator to measure performance. The system had 16 cores connected by NoC with 2x2x4 mesh (4 layers) topology. The network routers were 4-stage pipelined. Each link was 256 bits wide with four VCs, eight buffers per VC. Each layer had four tiles, each with an x86-64 in-order processor core, a router, and private L1/L2 caches with 16KB/256KB capacity. Cores and routers ran at a nominal frequency and voltage of (3GHz, 1.2V) and (1.8GHz, 1.2V), respectively. We used five operating points for cores and routers: (3GHz, 2.5GHz, 2GHz, 1.5GHz, 1GHz) for cores and (1.8GHz, 1.5GHz, 1.2GHz, 0.9GHz, 0.6GHz) for routers, with an assumption that the voltage scales linearly to the frequency. For each v/f transition, 2  $\mu$ s overhead was added. Constant latency of 45 ns was used for accessing DRAMs, which were connected via four off-chip channels at the top layer.

To estimate power consumption of cores and caches, McPAT [73] was used.

Table 3.2: Configuration Parameters

Thermal Model Parameters (from [101])			
Layer	Thickness ( $\mu$ m)	Thermal conductivity (W/(m K))	Specific Heat (J/(m <sup>3</sup> K))
Active Layer	150	100.0	$1.75 \cdot 10^6$
Interface	20	4.0	$4.00 \cdot 10^6$
Heatsink/Heat Spreader	7900	400.0	$3.55 \cdot 10^6$
$T_{amb}$	318.15 K		
$T_{max}$	348 K		

Due to lack of detailed network modeling in McPAT, DSENT [102], a network energy and area estimator, was used to model network power consumption. Regarding the DVFS support, the method in [103] was used to extend McPAT and DSENT. For 3D temperature modeling, Hotspot [101] with grid modeling was used. For all power modeling, 45nm process was used. To simulate different applications running together, we used 25 applications from the SPEC CPU2006 benchmark suite [104]. Eight random mixes were used as multi-programmed workload and each mix was simulated for 0.2 sec. We divided the mixes into two categories: mixes with high network loads (mixes 0-3), and mixes with low network loads (mixes 4-7). We omitted the list of applications for each mix due to the space limitation.

For the experiments, PI controller for cores similar to the ones used in many state-of-the-art DVFS techniques [90, 96] was designed and used in the baseline. We implemented the individual PI control method explained in Subsection § 3.5.1 and the method with weighted-power budgeting and performance/power estimation explained in Subsection § 3.5.2 and § 3.5.3. Also, to see the effectiveness of the hill-climbing algorithm, the method using exhaustive search instead of hill-climbing was added where the runtime overhead of the exhaustive search was excluded. Per-core thermal constraint of 348K was used. Emergency sleep state entrance for uncontrolled runtime variation was implemented for all policies. The DVFS control interval was set to 5 ms. Although thread migration could be considered for dynamic thermal management, we did not include the issue since it is not within the scope of this paper. However, our approach does not limit the use of any migration policy since it can be applied independently. In our experiments, we performed random migration with 100 ms interval.

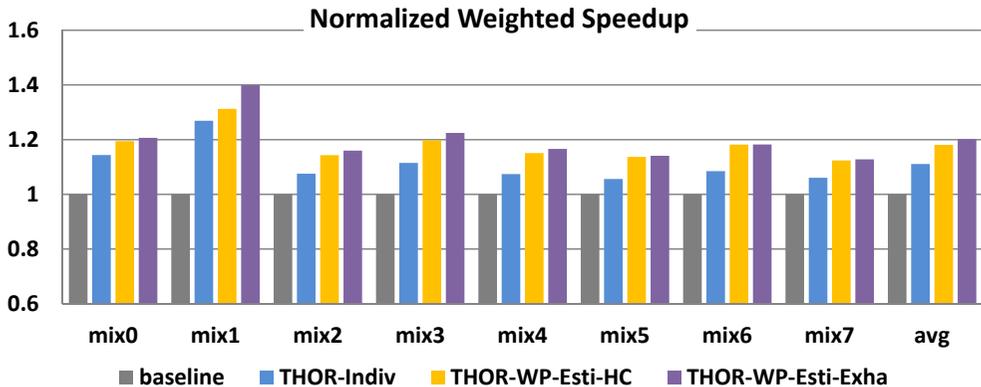


Figure 3.6: Weighted speedup of various approaches.

### 3.6.2 Overall Algorithm Performance

Figure 3.6 shows weighted speedups of the mixes for the algorithms under comparison, normalized against the baseline. In the figure, “THOR-Indiv” represents the individual PI control method explained in Subsection § 3.5.1. “THOR-WP-Esti-HC” represents the method explained in Subsections § 3.5.2 through § 3.5.4 which employs weighted-power budgeting, performance/power estimation, and hill-climbing algorithm. Finally, “THOR-WP-Esti-Exha” represents the method that performs exhaustive search to see the effectiveness of the hill-climbing algorithm.

The “THOR-Indiv” bars show that even for a simple individual PI control method, applying the network DVFS together can give a large speedup gain. On average, “THOR-Indiv” shows 11.1% better performance than the baseline.

The weakness of “THOR-Indiv” is at its unawareness of neighbor tiles. Tiles that are vertically adjacent have strong thermal correlation. Also, routers near the memory controllers incur more performance impact on the system throughput, because it is likely to become a traffic hotspot. However, “THOR-Indiv” is not able to exploit those properties and it just controls the operating

speeds according to its local temperature.

On the other hand, “THOR-WP-Esti-HC” is able to consider thermal correlation between tiles in a pillar by controlling the weighted-power budget. Also, the performance estimation model allows capturing the different performance/power impacts of routers according to their locations. Along with the help of performance/power estimation model, it can find proper operating points for the components. On average, “THOR-WP-Esti-HC” obtains 18.1% improvement over the baseline.

“THOR-WP-Esti-Exha” puts more effort on deciding the optimal operating points by performing exhaustive search over all possible v/f combinations. Because the purpose of the experiment with this method is to see the effectiveness of the hill-climbing algorithm, runtime of the exhaustive search was ignored. Despite the heavy computation effort, performance difference between “THOR-WP-Esti-Exha” and “THOR-WP-Esti-HC” was not significant. The differences for all mixes were within 2% except for mix1 that has difference of 6.5%.

The proposed methods are relatively more effective for memory-intensive mixes because they get more benefit from controlling router speeds. For applications mixes with high network loads, “THOR-WP-Esti-HC” shows improvement of 21.2% over the baseline and 14.8% for the mixes with low network loads. “THOR-Indiv” obtains 15.1% for mixes with high network loads and 6.9% for low network load mixes.

Currently, we do not consider the overhead of power regulators. Our preliminary results using 85% power conversion efficiency shows about 3.5% performance degradation.

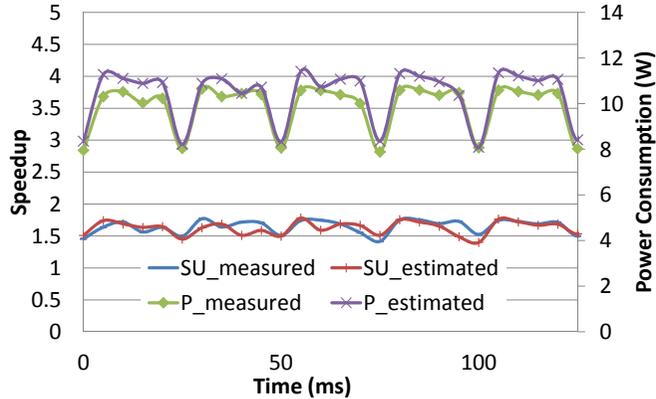


Figure 3.7: Accuracy of the estimation model.

### 3.6.3 Accuracy of the Estimation Model

Figure 3.7 shows the plot of measured/estimated value of speedup and power consumption captured from a tile, in the middle of running gcc. In the curves, we observe a cyclic behavior in both speedup and power consumption resulting from the attempt to meet the temperature constraint. Speedup curves of the figure show that our model accurately estimates the varying speedup and power consumption. The average errors of speedup and power consumption for all the applications were 4.9% and 5.3%, respectively.

### 3.6.4 Performance of the Frequency Assignment Algorithm

In this section, we compare the scalability of the exhaustive algorithm and the hill-climbing based algorithm for frequency assignment. We optimized both algorithms with a pruning technique. We tested the algorithms with various numbers of layers and operation points. Intel Nehalem i7 core running at 2.67GHz was used for the experiment. Figure 3.8 shows the runtimes of the two algorithms. X-axis represents the number of layers, which exponentially increases the number of possible solutions (also the complexity of the exhaustive algo-

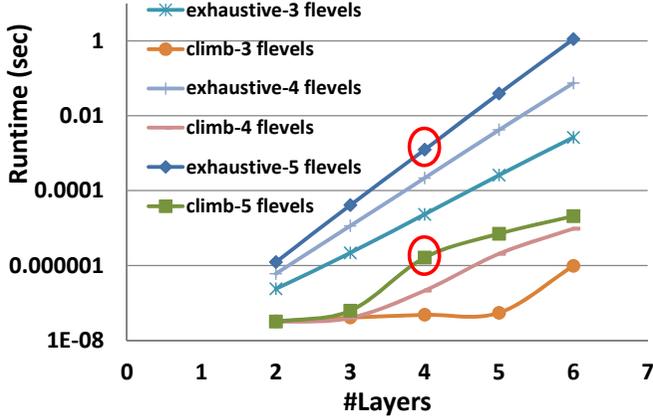


Figure 3.8: Runtimes of the frequency assignment algorithms.

rithm) calculated by  $(|F_c| \cdot |F_r|)^L$ . The Y-axis is the runtime of the algorithms drawn in log scale for visibility. The curves represent the runtime of the two algorithms, each with its own number of operating v/f levels (abbreviated as ‘flevels’ in the legend). The exhaustive method scales exponentially to the number of layers. The runtime of Hill-climbing (marked as ‘climb’ in the figure), on the other hand, is bounded above by  $((|F_c| + |F_r| - 2) \cdot 2L^2)$  as calculated in Subsection § 3.5.4. The two red circles point the parameters used in this paper (i.e., 4 layers and 5 operating levels). For the exhaustive search, it takes 1.2ms, which is 24% of the control interval. Considering the minimal speedup gained from the exhaustive search, this is not acceptable. Hill-climbing approach, on the other hand, takes only 1.6  $\mu$ s and it is only 0.032% of the control interval.

## Chapter 4

# Routing for Limited Bandwidth 3D NoC

### 4.1 Introduction

In this chapter, we propose a deflection routing for 3D networks with limited bandwidth vertical links. In the architecture we assume, the vertical links are serialized to mitigate the expensive cost of the TSVs. This can cause bottleneck problem at the vertical links, because such serialized links will incur long latency, and small amount of excess traffic can lead to a huge amount of queuing latency. We try to mitigate the problem by employing a deflection routing algorithm that efficiently splits the vertical traffic to the available vertical links in the layer. In the algorithm, the horizontal network uses a deflection scheme for destination conflicts, and vertical transitions are allowed only when there is an available vertical link. Unlike original deflection routing algorithms which trades energy efficiency with performance under 2D networks, our algorithm shows superior performance as well as energy consumption on 3D networks.

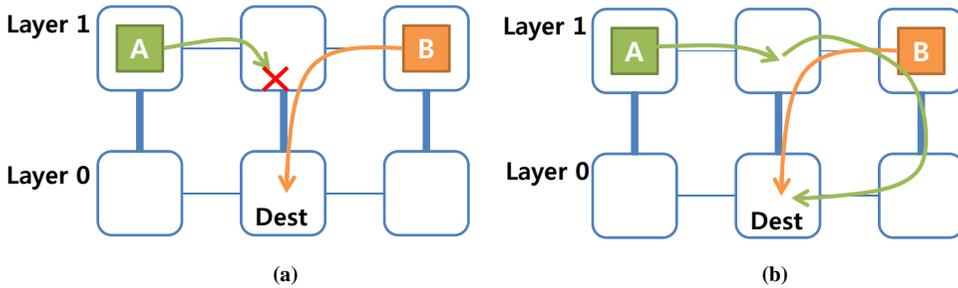


Figure 4.1: Motivational example.

## 4.2 Motivation

It is expected that applying TSV serialization scheme to 3D network routers will significantly drop the bisection bandwidth of the network. In other words, it is very likely that those TSV links will be highly congested and be the performance bottleneck. The problem is serious when static routing is used and the load to TSV links is unevenly distributed since the use of static routing will worsen the traffic condition in congested TSV links. As flits fill up a buffer in a router due to such a congested TSV link, backpressure starts to propagate from that TSV link and slows down the entire network. In this case, it may help mitigate the problem to take a detour and use other idle TSV links. Even though it may not render a minimal path, it can send a packet much faster than sending it through the congested minimal path.

The situation described above is shown in Figure 4.1 (a), where packets A and B are to be sent to Dest. When DOR (Dimension-Order Routing) is used, we cannot avoid the packets passing through the congested TSV link in the middle, while the other two TSV links remain idle. If one of the packets can be transferred through another TSV link as in Figure 4.1 (b), both latency and throughput of the network can be improved. In this context, deflection routing

seems to be an excellent solution. When two or more flits compete for a single TSV link, all flits other than the winning one will be automatically deflected to other directions, possibly looking for other available TSV links. Being deflected to another direction may incur longer latency in an ordinary network. However, in the existence of congested TSV links, the additional delay coming from detour can be smaller than the latency increase due to the congested TSV links. Thus, in addition to area and energy efficiency of bufferless deflection routing, we can also obtain higher performance.

deflection routing, we can also obtain higher performance. Alternatively, buffered adaptive routing can also be used to realize similar effect. However, its cost is generally very high. Complex architecture is needed because it needs to gather information on congestion in the neighborhood and it often demands a large number of virtual channels to avoid deadlock. If all aspects are optimally considered, adaptive routing can be made better than deflection routing in terms of hop latency. However, we will show that our deflection routing is much more efficient in terms of power consumption and the performance is comparable or even better under reasonable load. Note, however, that deflection routing can be considered as a case of low-cost adaptive routing.

### **4.3 Background**

In ordinary routers, there are buffers in each input port. When two or more packets compete for an output port, one of them wins and penetrates through the destined port. The remaining packets have to stay in the buffers and wait for the next cycle to compete again.

Bufferless deflection routing, on the other hand, does not have such buffers. When there is no congestion, the flits are statically routed in the same way as DOR. However, when a flit loses on arbitration, it is directed to any other port

(deflected) instead of being buffered (it actually uses router pipeline as buffers). Its performance is comparable to ordinary routing under low to medium load. However, as traffic load gets higher, the flits become more likely to be deflected and the network is easily congested. Thus the maximum throughput is degraded. The advantage is in its low complexity. Because this scheme does not require any buffers except for pipeline registers, it is simple and power efficient when traffic load is maintained under certain level. However, we will show later that, opposed to previous studies, this scheme can be made to actually improve performance as well as energy consumption, even under fairly high traffic load when used for 3D NoC with limited vertical bandwidth.

## 4.4 Related Work

Bless [105] was the first work that eliminated buffers in NoC and lead to power and area efficiency by using bufferless deflection routing. A number of approaches have been proposed later to increase efficiency of bufferless deflection routing. In CHIPPER [44], a butterfly permutation network comprised of four 2x2 switches is used instead of a crossbar to reduce area and power overhead. The permutation network is fully connected, but is a blocking network and thus only supports partial permutability. However, since deflection network does not always need full permutability, it incurs very little performance overhead. Also, injection/ejection ports are handled specially. That is, instead of going through the permutation network, a locally destined flit is ejected in an early stage. When a flit is to be injected, the injection logic looks for an input port that is made free either by no input or ejection. Only when a free slot is found, the injection logic injects the flit into the slot.

MinBD [106] proposes the use of a small side buffer inside a router. Instead of deflecting a flit, they put it into the side buffer when possible and re-inject it

when there is a free slot in the router. To avoid starvation, there is a maximum threshold for number of cycles that a flit can stay at the head of the buffer. If a flit at the head stays longer than the threshold, it is swapped with a random flit in the router. This scheme helps reducing deflection rate and increases network performance.

Recently, a hierarchical ring based NoC which also utilizes deflection routing has been proposed [107]. In the work, ring topology is combined with deflection routing to build up more power-efficient and low-cost router architecture. To acquire scalability for rings, hierarchical ring topology where bridge routers are used to connect between different levels is proposed.

There are a number of previous publications that dealt with similar problems of limited bandwidth in 3D network. In Rahmani et al. [32] and Rahmani et al. [33], routing algorithms to avoid 3D bus congestion were proposed. The architecture they assumed is different from ours in that they used a 3D bus hybrid architecture. Recently, a kind of adaptive routing was proposed to split the traffic load to TSVs [108].

All these approaches are based on adaptive routing approach. As mentioned in the previous section, although they can achieve the primary goal, they increase router complexity and consume more power because the routers need to gather additional information. Additionally, adaptive routing often requires large number of virtual channels to avoid deadlock, adding even more complexity.

## **4.5 3D Deflection Routing**

### **4.5.1 Serialized TSV Model**

In its simplest model, a serialized TSV link is composed of a transmit buffer, a set of TSVs and a receive buffer. When a flit is to be transmitted to the other

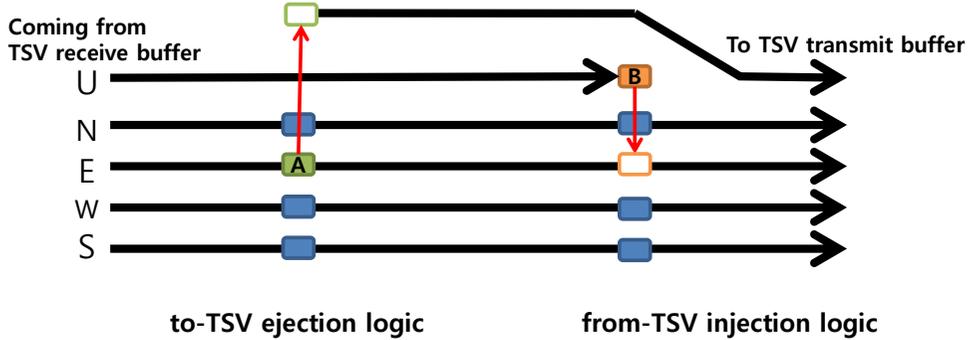


Figure 4.2: An example of TSV injection/ejection scheme.

side, it is first put into the transmit buffer. After the transmit buffer is filled, the TSV starts sending the flit part by part. When the whole flit has been sent after a few cycles, the flit reconstructed in the receive buffer becomes ready to leave the TSV link. To avoid deadlock, additional flits should not be put into the transmit buffer until the received flit leaves the link and the receive buffer is emptied (the reason will be explained in § 4.5.3). In other words, **each TSV link should have at most one flit at a time** and this is the condition that is maintained to avoid deadlock. To maintain the condition, the TSV link is *closed/open* when the receive buffer is full/empty. This control can be done with the overhead of adding a single TSV for sending information on open/closed status from the receiver side to the transmitter side, which is required regardless of the routing algorithm.

In this work, we assume synchronous serialization. In other words, when serialization ratio is  $n:1$ , it takes  $n$  network cycles to transmit one flit. Asynchronous serialization may allow more area saving or higher bandwidth, but it needs additional overheads such as separate oscillator. Nevertheless, our work can be easily extended to asynchronous serialization without any other consideration as long as open/closed status of the TSV link is known to the router.

## 4.5.2 TSV Link Injection/ejection Scheme

Our routing algorithm and architecture are based on MinBD [106] as shown in Figure 4.5. The detailed explanation will be given later in § 4.5.5.

In applying deflection routing to 3D network with serialized TSV links, there is a directly faced problem called *excess input problem*. The problem occurs when the number of incoming flits exceeds that of available output ports. This comes from the serialization of TSV links. While the crossbar switch in a router can send one flit through a TSV link every cycle, the TSV link cannot emit flits at the same rate. Consequently, an inter-layer output port can be blocked at a certain time point. At that moment, a flit that gets deflected to the blocked port cannot be routed, disabling the function of deflection routing. This means that vertical ports connected to TSV links have to be handled differently from planar (2D) ports. Note that this problem cannot be settled by having additional buffers at the vertical output ports. No matter how many buffers there are in a vertical port, the maximum input rate (one flit per cycle) exceeds the maximum output rate (one flit per  $n$  cycles where  $n$  is the serialization ratio) through the TSV link. Then eventually the buffer will become full, blocking the port.

One solution is allowing flit injection from an incoming TSV link only when the paired outgoing TSV link is open so that a flit can exit through that link. In this way, the number of available outgoing ports can be always made equal to or greater than the number of incoming flits, so that the deflection routing mechanism always works. Another solution is treating ports connected to TSV links (up/down direction) in a special way as follows. Once a TSV link has received a flit into the receive buffer, it looks for a free slot so that the flit can leave the receive buffer just like an injection port (from-TSV injection). On the other hand, when a flit is destined to a TSV link is found, it is always

put into that TSV link just like an ejection port if the TSV link is open (to-TSV ejection). If the link is closed, then the flit should be routed to a different direction. If multiple such flits are destined to the same direction, then only one flit is selected and all others have to be routed to different directions. This scheme is illustrated in Figure 4.2. Assume that four flits come from each of the four 2D directions. Among them, flit 'A' coming from the east input port is directed to vertical direction. The to-TSV ejection logic detects this and puts it to the TSV transmit buffer, so that it could be transmitted to another layer by TSV link. At the same cycle, the TSV receive buffer is filled up with flit 'B', which is ready to be injected. When the from-TSV logic tries to find a free slot in the 2D ports, we have the east port empty because the flit entered from east port has just been ejected to the vertical port. It works similarly when a from-TSV flit wants to be ejected to the local node. If it cannot be ejected to the local node due to busy local ejection port, it is deflected<sup>1</sup> (i.e., injected) to a 2D direction if a free slot is found in that direction. Although we could choose to wait at the TSV receive buffer until the ejection port becomes available, we thought deflecting to a 2D direction (instead of waiting) would utilize the vertical links better, and this scheme fits well with the deadlock avoidance protocol, which will be explained later in § 4.5.3. If even 2D ports have no free slot, then the from-TSV flit stays stalled. This solution effectively solves the excess input problem because only planar ports participate in the deflection (i.e., there are always four inputs and four outputs).

The first solution may have deflections to TSV link, which can degrade the performance (*deflection-to-TSV problem*). The key point of applying deflection routing is to avoid crowded TSV links. However, deflecting a flit to a TSV link actually adds redundant traffic overhead to the performance bottleneck. While

---

<sup>1</sup>Alternatively, the flit can be put into the side buffer (refer to § 4.5.4)

a flit is being sent through a TSV link uselessly, other flits are prohibited from using the link. The traffic overhead is actually doubled because the packet has to go backward eventually<sup>2</sup>. Of course, the flit itself will suffer from additional two-hop latency going through the TSV links and router pipelines. In addition, TSV links consume more energy than ordinary links [109] and thus the energy penalty is larger.

Thinking of the deflection-to-TSV problem, the first solution to the excess input problem may not be ideal. On the other hand, the second solution using TSV injection/ejection scheme has no deflection to TSV link in normal situations. Actually, there is some case where deflection to TSV must be allowed to avoid deadlock (see § 4.5.3). However, such a case is very rare and does not affect the performance so much. Comparison of the two solutions is given in § 4.6. Since the second solution always gives better results in terms of performance and energy consumption, we take the second solution in our approach. Proposing the second solution together with additional mechanisms for deadlock and livelock avoidance is our main contribution in this paper.

### 4.5.3 Deadlock Avoidance

Original deflection routing is deadlock free because no flit is ever stuck or waiting for other resources. However, because of the TSV link ejection/injection scheme, a deadlock may occur combined with livelock. In the example shown in Figure 4.3, it is assumed that there are two layers of planar NoCs and only one router connecting them. Layer 0 is full of flits whose destinations are all on layer 1. Conversely, layer 1 is full of flits whose destinations are all on layer 0. Also assume that all the TSV links are occupied, waiting for free slots to come

---

<sup>2</sup>Since we are basically taking ZYX routing, deflection to a TSV link (in Z direction) means deflection to the unwanted direction from the destination. Thus the flit should go backward later to reach the destination.

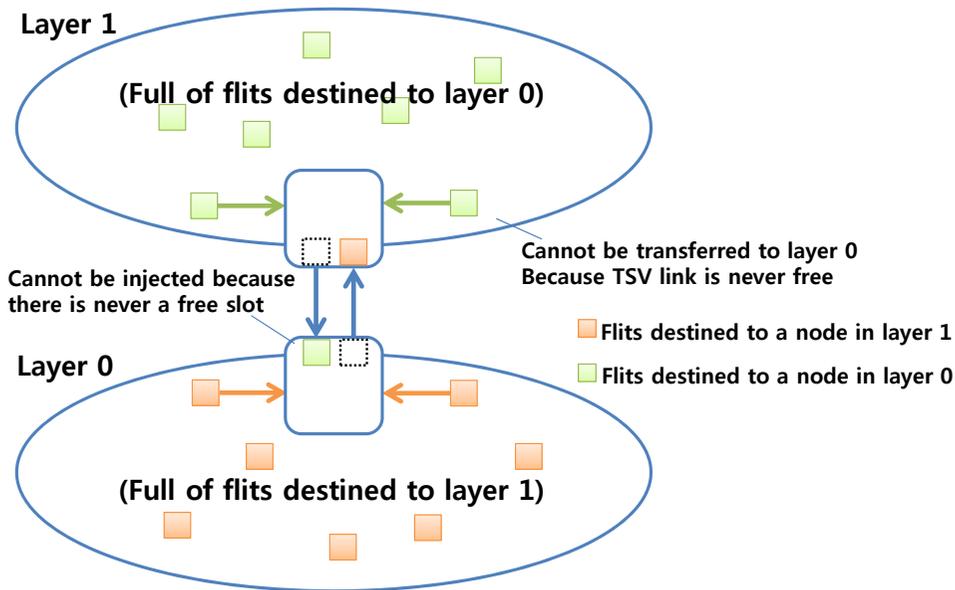


Figure 4.3: Flits in TSVs are in deadlock state while flits in 2D layer experience livelock.

up for injection of the flits in the receive buffers. Unfortunately, no free slot is available because the network is full of flits, and thus no TSV injection occurs, and in turn, no TSV ejection can occur since TSV links are closed. As a result, the flits inside the TSV receive buffers suffer from a deadlock, while all the flits in the 2D networks experience livelock.

The situation is drawn in more detail in Figure 4.4 (a). For simplicity, let us assume that there are only two planar ports instead of four. In the figure, the TSV receive buffers are full and there is no free slot. If there is continuously no free slot for further cycles, it is deadlock. Here, because the transmit buffers of the TSV links are empty (see the condition in § 4.5.1), we can perform an escape action as in Figure 4.4 (b) to avoid the deadlock. In the escape action, a flit is put into the transmit buffer of the TSV link even though the receive buffer of the link is full. Then the flit in the receive buffer can be injected using the free

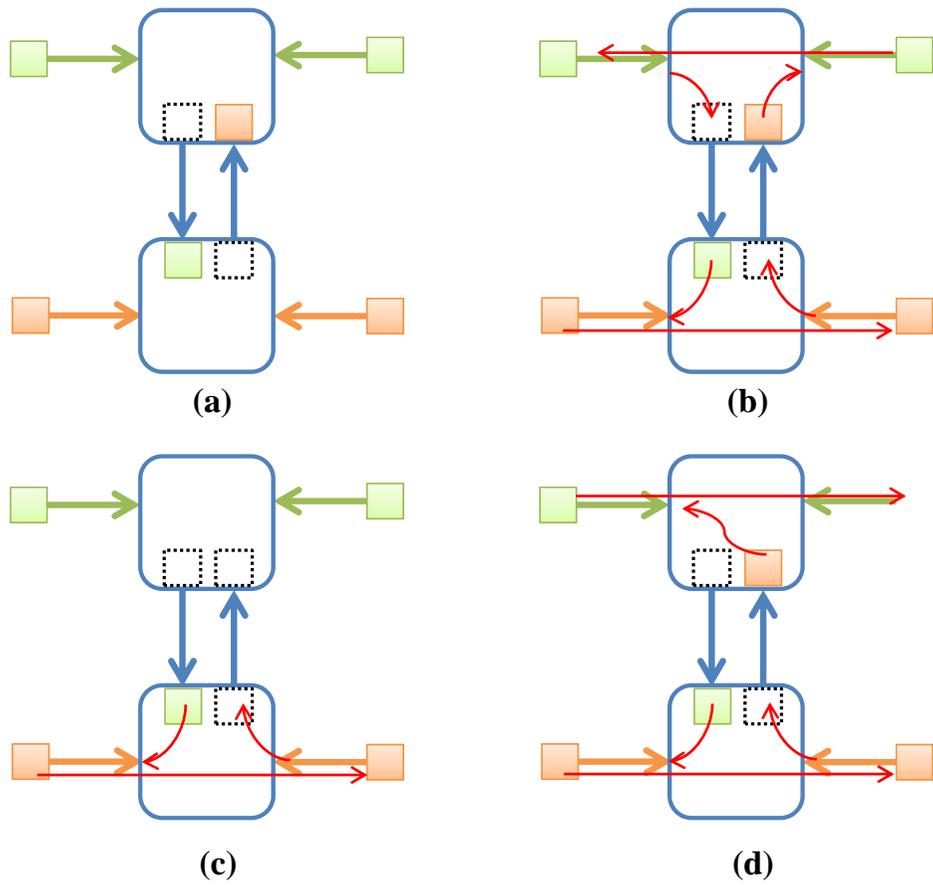


Figure 4.4: Performing escape action to avoid deadlock.

slot made by that action. In other words, the ports to TSV links participate in the deflection for that cycle. If the action is performed in both routers, we can escape from the deadlock and the condition in § 4.5.1 still holds.

A router performs the escape action when the following three conditions are met within the router:

1. There is no free slot in the 2D input ports.
2. The TSV receive buffer is ready to inject (full).
3. The TSV transmit buffer is empty.

Once the escape actions are taken on both layers, then the flits in the receive buffers will be emptied at the same cycle, escaping from the deadlock situation. Note that all conditions are based on the router's local information and the scheme does not require additional physical connections or extra traffic. Sometimes, the three conditions can be met on only one layer (say, router 0), which is certainly not a deadlock. Because routers decide to take the escape action based only on its local information, it is possible that only one side of the router (router 0) takes the escape action while the other (router 1) does not. Still, it does not break the functionality of the routing scheme. This is depicted in Figure 4.4 (c) and (d). Consider the case that the actions are taken only on router 0. This implies either

1. There is no flit in the receive buffer of router 1 (as in Figure 4.4 (c)) or
2. There is a free slot on router 1 and the flit in the receive buffer gets injected (as in Figure 4.4 (d)).

In any of the two cases, filling up the transmit buffer on layer 0 does not cause any trouble since the receive buffer on the other side of the TSV link is empty or will be emptied.

Note that the transmit buffers are filled up only when the receive buffers are empty or guaranteed to be emptied, which is the key to avoid deadlock. The deadlock problem cannot be solved by just adding more buffers. Regardless of the buffer size, the buffer will become full because of higher input rate than output rate.

#### 4.5.4 Livelock Avoidance

There are numerous ways to avoid livelock. One easy way is to prioritize old flits [105]. However, it was claimed to be ineffective [44, 110] for its long critical path of the sorting circuit. Instead, Golden Packet [44] rule was introduced to ensure livelock freedom. According to the rule, each router keeps a tuple (sender, transaction ID), where transaction ID is a unique identifier of a packet from one sender node, such as MSHR number. The tuple rotates every  $L$  cycles in static order, where  $L$  is the maximum zero-load latency of a packet. The packet that matches the tuple becomes the golden packet. During the  $L$  cycles, the flit with the golden packet ID wins on all conflicts. Because of this, the flit is guaranteed to arrive at the destination within  $L$  cycles once it becomes the golden packet, preventing livelock in a cheap way.

The golden packet rule needs some modification to be applied to our work. First, in our architecture, a golden flit may not be routed toward its minimal direction because of a closed TSV link. This problem can be easily fixed by using side buffers. When a flit (can be a golden flit) is to be routed to a TSV link that is closed, the flit is put into the side buffer instead of being deflected. If the flit at the head of the side buffer is a golden flit, it is swapped with a flit (if it exists) destined to the TSV link as soon as the link becomes open. Even if it is not a golden flit, it is forced to be injected within pre-defined threshold cycles as in MinBD [106]. Then we can calculate the maximum latency of a

golden packet using the following terms, and thus ensure livelock freedom.

- $P$ : maximum size of a packet.
- $L_{hop}$ : latency for a flit to traverse one hop through a planar link.
- $L_{ser}$ : additional latency for a flit to transfer through a serialized TSV link.
- $S$ : size of the side buffer in number of flits.
- $T$ : threshold of a flit to stay at the head of a side buffer.  $T$  should be equal to or larger than  $L_{ser}$  to avoid injection failure after  $T$ .
- $N_h$ : number of horizontal (planar) hops between farthest source-destination pair.
- $N_v$ : number of vertical (TSV) hops between farthest source-destination pair.

When a golden flit traverses one hop through a planar link, it takes  $L_{hop}$  cycles. Traversing through a vertical link is a little more complex. If the TSV link is open, it only takes  $L_{hop} + L_{ser}$  cycles. However, as explained, the golden flit has to be stored in the side buffer if the TSV link is closed. Dequeuing all flits ahead of the golden flit from the side buffer takes at most  $(S-1)$  times  $T$  cycles. Once the golden flit is moved to the head, it waits there no longer than  $L_{ser}$  cycles since the TSV link will become open within that period. Thus the maximum vertical hop latency becomes  $L_{hop} + 2L_{ser} + (S-1)$  times  $T$ . Because golden flits always take a minimal path, the maximum latency of a golden flit is  $N_h \times L_{hop} + N_v \times \text{vertical\_hop\_latency}$ . Another thing we have to consider is that there can be multiple golden flits from a single golden packet. Multiple golden flits can collide at a router, because of waiting in the side buffer and/or being

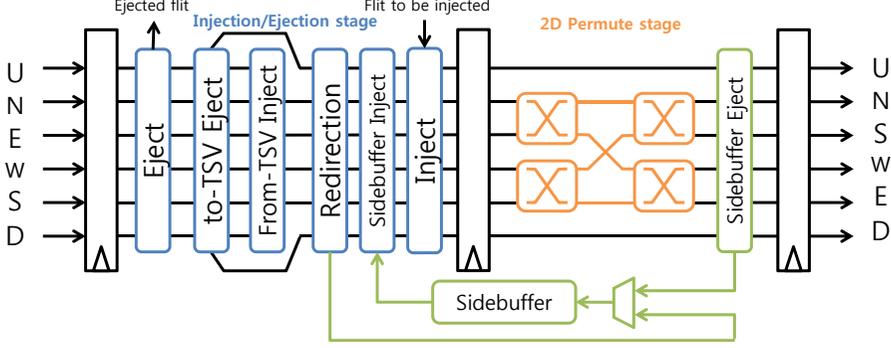


Figure 4.5: Proposed router architecture.

deflected prior to becoming golden. When this happens, the sequence number of the flit is used to prioritize one flit over others. Once deflected, the flits waste at most two hop latencies, and a golden flit can be deflected at most  $P-1$  times. Considering this, the maximum golden packet latency  $L$  can be calculated as follows:

$$L = N_h \times L_{hop} + N_v \times (L_{hop} + 2L_{ser} + (S - 1) \times T) + 2 \times L_{hop} \times (P - 1) \quad (4.1)$$

#### 4.5.5 Router Architecture: Putting It All Together

Considering the routing scheme described above, our proposed router architecture is shown in Figure 4.5. It is based on [106], and the only difference is the number of ports and the TSV ejection/injection logic. The components in the first stage are placed in the priority order of the operations. Ejection logic is placed first. TSV ejection/injection modules come next to prioritize TSV links. Side buffer redirection/injection modules are the next. Injection logic is placed at the end of the stage. This is because too much injection will degrade the performance and it's better to inject when a free slot exists after all the

steps. Second stage consists of 4x4 permutation network and side buffer logic. Note that (N, E), (W, S) pairs are changed to (N, S), (W, E) pairs [44, 106] to minimize deflection. Pseudo-code of overall routing algorithm is shown in Figure 4.6.

#### 4.5.6 System Level Consideration

There are a few issues to be addressed in system level design, although they are not part of the router design. First, because deflection routing does not guarantee the flits within packets to arrive at the destination in the designated order, a reordering buffer is required. Second problem is message-level deadlock (namely, protocol deadlock). Even when using deadlock-free routing algorithm, dependencies of messages can form a deadlock configuration. Often, it is handled by separating messages into different networks, using physically independent networks or virtual channels [111] for its less cost. In deflection routing, however, virtual channels cannot be implemented because it would force flits to be blocked at some point. One way is to use system-wide time multiplexing, but it would degrade performance too much and thus unacceptable. For those problems, we adopt solutions from [44].

Reordering buffer problem can be solved by using MSHRs as buffers. In modern processors, MSHRs already exist to support non-blocking cache. Because an MSHR entry is reserved when a new request is issued, it can also serve as reordering buffers. Similarly, request buffers existing in L2 caches for coherence protocols can be used for the same purpose.

In deflection routing, a local injection port can be blocked when no free slot is available due to completely full network. That may block processing the packet coming into the local ejection port, thereby taking up receive buffer space. When there is no more receive buffer space to allocate at the endnode, message-level

```

1  /* Ejection */
2  if (flits destined to local endnode are found)
3      Eject one flit to the endnode
4
5  /* to-TSV ejection */
6  if (the router is in escape condition) {
7      for each TSV ports {
8          if (flits directed to the TSV port are found)
9              Put one of the flits into the TSV port
10         else
11             Put a random flit into the TSV port
12     }
13 }
14 else if (flits directed to TSV links are found and the TSV port is open)
15     Put at most one such flit into each TSV port
16
17 /* from-TSV injection */
18 for each TSV ports {
19     if (the TSV port has a flit to inject)
20     if (the router is in escape condition or a free slot is found)
21         Inject the flit
22 }
23
24 /* Side buffer redirection & injection */
25 if (side buffer has a flit) {
26     if (a free slot is found)
27         Inject the flit at the head of the side buffer
28     else if (flit at the head of side buffer reached the threshold)
29         Swap the flit at the head with a random flit
30 }
31 if (flits to be injected from the endnode exist and a free slot is found)
32     Inject the flit from the endnode
33
34 /* permutation */
35 Input flits in planar ports into the permutation network
36
37 /* Side buffer ejection */
38 if (flits to be deflected are found and the side buffer is not full)
39     Enqueue one such flit into the side buffer

```

Figure 4.6: Pseudo-code of overall routing algorithm

deadlock occurs. Such a deadlock problem is solved using retransmit-once protocol. The protocol prevents this from occurring by discarding the arrived packets and letting the sender re-transmit. To prevent repetitive retransmissions, the receiver keeps a retransmit-request queue, which is implemented using bit-vector with one bit per MSHR of each sender. Later, when request buffer becomes available, the receiver allocates buffer space for one of the entries in the queue, and sends retransmit packet to the corresponding sender. When the sender retransmits, since the buffer space has already been allocated, no more retransmission is needed.

## 4.6 Experimental Results

### 4.6.1 Experimental Setup

To obtain network performance measure, we used an in-house, cycle-accurate simulator. To show the effectiveness of our router, we compared it with three existing routers: a simple buffered router without virtual channel, a buffered router with four virtual channels, and a minimal adaptive router. A naive deflection router which performs some unnecessary TSV deflections (the first solution in § 4.5.2) was also evaluated. For minimal adaptive router, we modified adaptiveXYZ, which has been originally designed for bus-hybrid 3D NoC in [32], to be used in our architecture. The design parameters of each router are shown in Table 4.1, where 4x4x4 mesh architecture with 128 bits for each unidirectional link is assumed. Since 4:1 serialization is used for vertical links, a total of 1024 vertical links (TSVs) exists between adjacent layers (thus 2048 TSVs for layers in the middle). Simple DOR is used for non-adaptive buffered routers. Deflection routers also use the same DOR for initial routing of flit, which can be deflected depending on the situation.

To obtain numbers for power consumption, we modified DSENT [102] (which

is the latest version of Orion 2.0 [71]), a network energy estimator, and integrated it into our simulator. Router components as well as wires and TSVs were modeled under 45nm technology for simulation (energy models for TSVs are from Ouyang et al. [112]). To be a fair comparison, buffer bypassing technique was taken into account in all buffered routers. Also, increased flit-width overhead of additional information in deflection routing was also considered when calculating energy consumption and area cost.

We tested the networks with four different traffic patterns and a NoC benchmark. First, a uniform random traffic pattern is used as the basic traffic pattern. All nodes send packets to all other nodes at equal probability. Hotspot random traffic is similar to uniform random traffic, but 10% of total packets are destined to a ‘hotspot’ of the network. Permutation traffic, on the other hand, does not send packets to random nodes. Instead, each node has only one destination, and each node receives packets from exactly one source node. Two kinds of permutation traffic patterns are used: bit-complementary and tornado. Also, to capture the usefulness of our scheme in more realistic situation, a NoC communication task graph suite called MCSL benchmark [113] is used.

To gather experimental results for real application workload, memory access traces of 21 applications obtained from SPLASH-2 [114] and PARSEC [115] benchmarks are used. Sniper multi-core simulator [116] is used to obtain memory traces of one billion instructions for each applications and it is fed into the network simulator. In the system, each node has an x86 Nehalem-like processor, a private L1 cache, and a slice of distributed shared L2 cache. Low order bits of cache line address determine the slice that has the line as in S-NUCA [117] system. We modeled the coherent cache system using MESI protocol and included the effect on the stall time due to memory access (e.g., round-trip latency) and energy consumption of the cache coherence protocol.

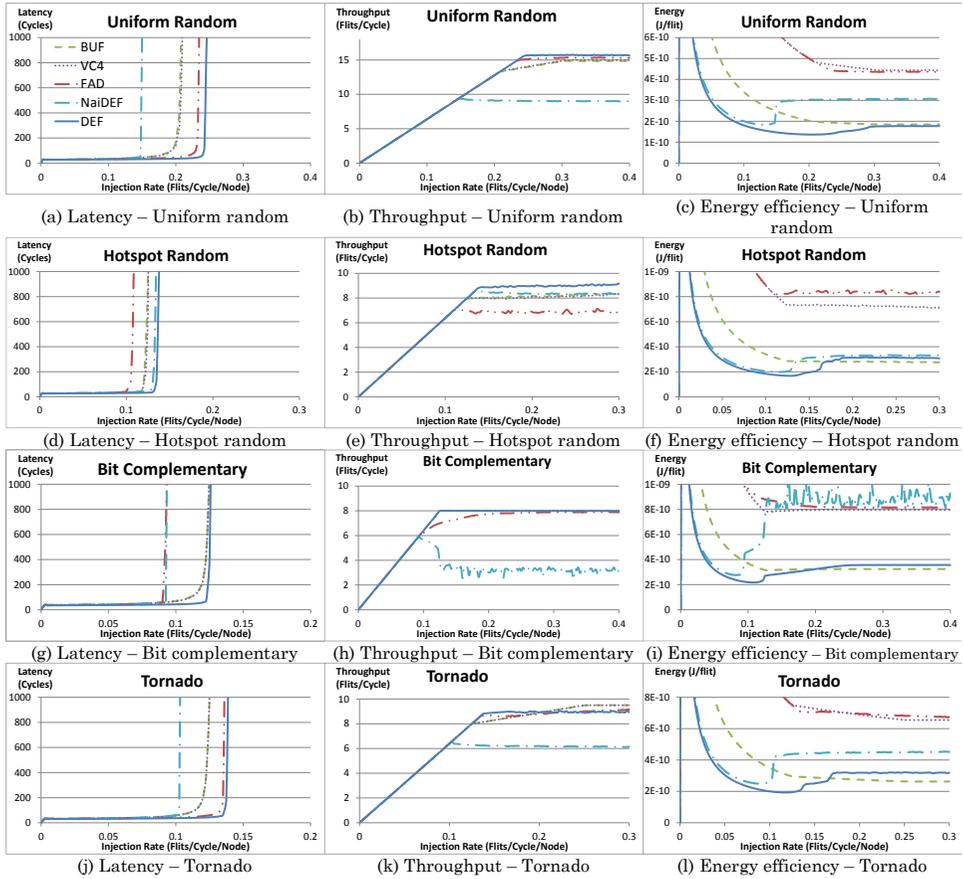


Figure 4.7: Experimental results under synthetic traffics. Ranges are adjusted for visibility.

#### 4.6.2 Results on Synthetic Traffic Patterns

Figure 4.7 shows the graphs for experimental results for synthetic traffic patterns. In the legend, “BUF”, “VC4”, “FAD”, “naive-DEF”, and “DEF” mean simple buffered network, buffered virtual channel network, adaptiveXYZ, naive implementation of deflection network, and the proposed deflection network, respectively. Four traffic patterns are tested with each packet having size of 4 flits.

Figure 4.7 (a), (d), (g) and (j) show the injection rate-average latency graph under different traffic patterns. As can be seen, saturation point of our algorithm is higher than others. It is even higher than adaptiveXYZ algorithm. Roughly defining saturation load as the load level where the average latency exceeds 500 cycles, our network performs 10.4% better than simple buffered network and 17.0% better than adaptiveXYZ in geometric mean. This comes from two reasons. First, adaptiveXYZ algorithm is a kind of minimal adaptive routing. Even though it tries to find available TSV links adaptively, it cannot utilize the ones that do not fall into the bounding box of the source and the destination. While flits of other algorithms have to wait in the buffer on minimal paths, flits of deflection routing can advance to the positions near the destination. Second reason comes from separable virtual channel allocator. Especially when congestion is high, separable allocator often fails to pull out the optimal choice and crossbar utilization becomes low. This is why the performance benefit of our algorithm is more significant in the presence of hotspot.

Figure 4.7 (b), (e), (h) and (k) are injection rate - aggregate throughput graph. TSV link utilizations, the most significant bottleneck, will eventually become full when the traffic load gets higher. Thus saturated throughput does not show much difference. By geometric mean, our network is 2.3% better than simple buffered network and 6.2% better than adaptiveXYZ. It is rather astonishing to find that the adaptive routing performs worse than simple buffered router. However, as can be found out from the graph, the performance of adaptiveXYZ is better than the simple buffered router in all traffic patterns except for hotspot random traffic pattern. When hotspot traffic pattern is excluded, performance of adaptiveXYZ becomes about 1% higher than proposed router, and that of simple router becomes almost same as that of the proposed router. This result says that when a hotspot exists inside the network, carelessly adopting adap-

tive routing designed for other architectures can only cause harm instead of benefit. Another aspect to be paid attention is that our algorithm reaches its maximum bandwidth quickly than other buffered routers in all traffic patterns. All other buffered routers need to stress the network much further to reach the full throughput. Thus we claim that our network is more likely to show higher throughput under a reasonable traffic condition. Naive implementation of the deflection routing, on the other hand, shows very poor bandwidth in most cases. As injection rate gets higher, it starts deflecting flits through TSVs, incurring significant overhead both on bandwidth and latency. Our network shows 65.1% higher throughput than naive deflection network.

Previous papers on deflection routing emphasized their superior power efficiency. Our algorithm also shows very good energy consumption per flit and it is shown in Figure 4.7 (c), (f), (i) and (l). Deflection router consumes less static power for its smaller permutation network structure and absence of buffers. Also, dynamic energy needed per hop is less because it does not need buffer read/write energy. This is why our routing has excellent energy efficiency over others. Its energy consumption becomes the same or higher than single channel buffered router on high load because of larger dynamic energy consumed by larger hop counts. However, it is better than any other routers and still better than single channel buffered router under low to medium load. When we measure the minimum energy consumption per packet of our deflection network, it is 30.8% lower than simple buffered network, and 73.1% lower than adaptiveXYZ.

Even when the number of deflections starts to rise, saturated energy consumption per packet of our network is 57.6% better than adaptiveXYZ and only 9.1% higher than simple buffered network. Naive deflection router shows fair energy efficiency when the load is low, but it steeply gets higher from the

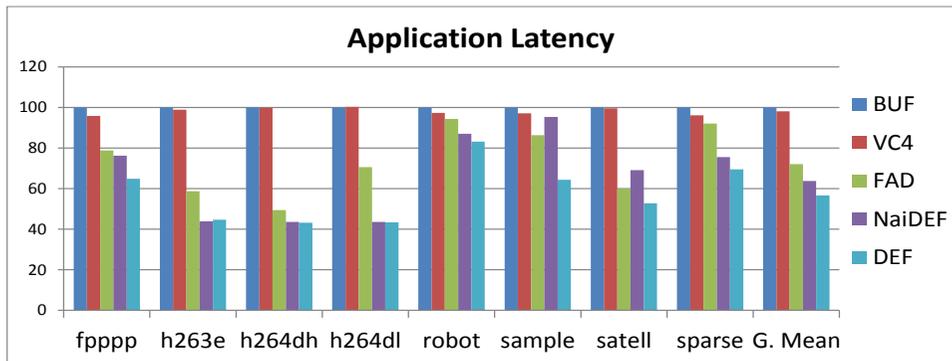


Figure 4.8: Application latencies of MCSL on different networks.

point where it starts TSV link deflections. At saturation, it goes as bad as the expensive adaptiveXYZ. Our deflection router also has possibility of unwanted TSV link deflections, but TSV injection/ejection scheme prevents the number of TSV link deflections from overwhelming. Thus the energy efficiency is maintained at a reasonable level even at a very high traffic load. Virtual channel router and adaptive router suffer from large static as well as dynamic energy coming from more buffers and virtual channels and thus their energy efficiency is poor.

### 4.6.3 Results on Realistic Traffic Patterns

To capture performance statistics on more realistic traffics, each network was run under a set of application traces where tasks execute instantly. Eight application traces were adopted from MCSL benchmark. The applications were run for ten iterations and total execution latency as well as energy consumption was measured. Table 4.2 shows their statistics.

Figure 4.8 shows latencies taken to run the applications and their geometric means. All results were normalized to the latency of simple buffered router network. In all applications, proposed deflection router outperforms other routers.

Table 4.1: System Design Parameters

Network	4 x 4 x 4 Mesh, 128-bit flit width, limited vertical bandwidth, 1-cycle link latency	
Core Model	Out-of-order x86, dispatch width 4, 128-entry instruction window	
L1 Cache	Private, 16KB 4-way I-Cache and 16KB 8-way D-Cache, 16 MSHRs	
L2 Cache	Shared, perfect, 16 request buffers per slice	
Routers	Simple Buffered	No VC, buffer bypass, buffer depth 8, 2-cycle latency
	Buffered, Virtual Channel	4 VCs, buffer bypass, buffer depth 8, 2-cycle latency
	AdaptiveXYZ	
	Naive deflection	No VC, 8 side buffers, 2-cycle latency, retransmit-once protocol
	Deflection (proposed)	

Table 4.2: MCSL Application Statistics

Application	Tasks	Edges
fpppp	334	1145
h263e	201	296
h264dh	6343	9509
h264dl	51	71
robot	88	131
sample	612	1021
satell	4515	7104
sparse	96	67

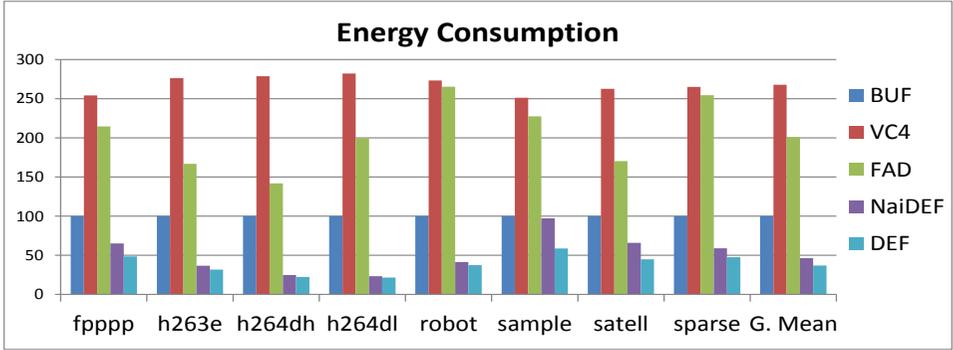


Figure 4.9: Energy consumptions of MCSL on different networks.

In geometric mean, Deflection routing performs 43.3% better than buffered routing and 21.3% better than adaptiveXYZ. Even without the TSV injection/ejection scheme (naive deflection routing), it performs better than other routers. This mainly comes from the fact that most of the applications have relatively low traffic load. As can be seen from § 4.6.2, naive deflection router performs comparable to deflection router under low injection load. In those applications, while some tasks emit messages, the other tasks have to wait until their dependencies have been resolved, so congestion remains under certain point. Because of this, naive deflection router performs almost same as deflection router in many applications. However, the difference between the proposed router and naive deflection router becomes great on applications with large number of edges, such as *satell* or *sample*. *H264dh* is an exception. Even though it also has a large number of edges, many of the edges have the same source or destination and sizes of messages are very small, so that the actual traffic load is actually low, and thus the difference between naive and proposed routers is not so large. In geometric mean, the proposed deflection router is 11.2% better than naive deflection router.

Energy consumption, which shows more gap between different routers is

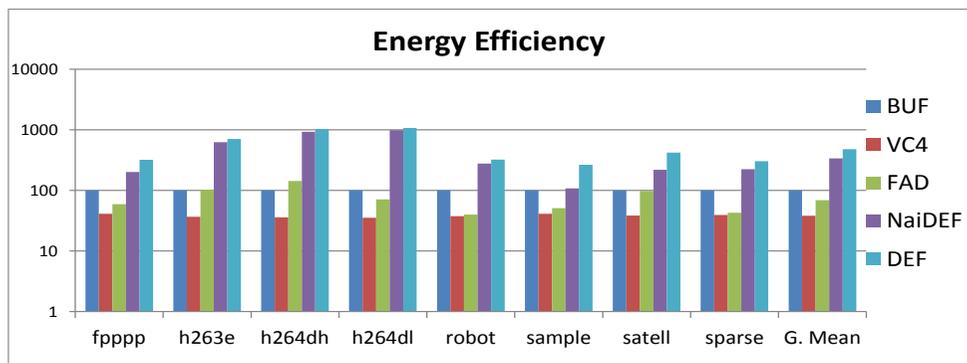


Figure 4.10: Energy efficiency of MCSL on different networks.

shown in Figure 4.9. Again, the proposed deflection router performs the best among all routers. The most power-hungry router was VC4, buffered router with four virtual channels. Because of its large number of buffers and the virtual channel allocator, it shows more energy consumption than the others. AdaptiveXYZ also suffers from large energy consumption coming from virtual channels, but less than VC4 because of less static energy owing to shorter execution latency. Through elimination of buffers and efficient control mechanism, the proposed deflection router shows the smallest energy consumption. On setting simple buffered router as the baseline, it saves 63.0% of energy in geometric mean. Compared to VC4, it consumes about 86.2% less energy consumption.

Another metric that can be used to evaluate a design is looking at its energy efficiency. One way to do it is using inverse of EDP (Energy-Delay-Product). Figure 4.10 shows the comparison on this metric. Because the proposed deflection routing performs better on both latency and energy consumption, energy efficiency shows significant difference from simple buffered router, sometimes more than 10 times. VC4 is the network with worst energy efficiency. As in Figure 4.8 and Figure 4.9, it consumes tremendous energy, but gains very little performance benefit from it, resulting in poor energy efficiency. AdaptiveXYZ

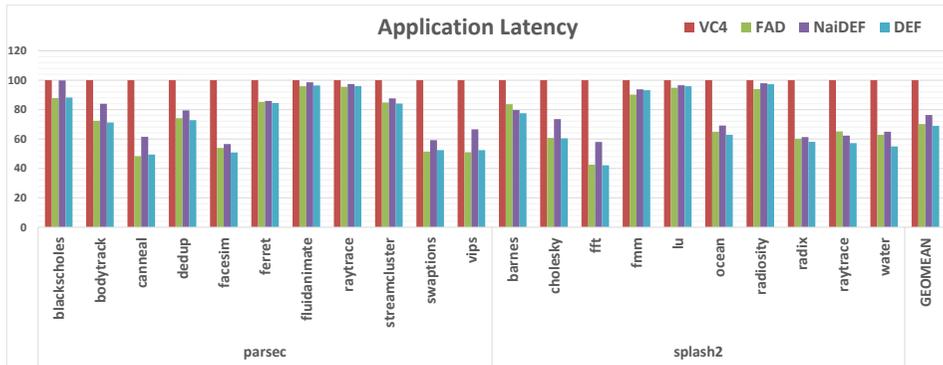


Figure 4.11: Application latencies on different networks.

performs better, but also worse than simple buffered router because its performance benefit cannot offset the excess energy consumed by virtual channels and their buffers. The proposed deflection router shows 4.8 times better energy efficiency than the simple buffered router and 12.5 times better than VC4. Naive deflection routing also shows good energy efficiency, but the proposed routing shows 41.4% better efficiency.

#### 4.6.4 Results on Real Application Benchmarks

To see the effectiveness of our network used for a real system, we used benchmark applications in SPLASH-2 [114] and PARSEC [115]. As was done in § 4.6.3, we collected energy consumption as well as application latency. In this experiment, however, simple buffered router is excluded because the cache coherence protocol requires virtual channels to avoid protocol deadlock. Even though deflection routing does not have virtual channels, it does not suffer from any protocol deadlock, which is another advantage of deflection routing. Also, the overhead of maintaining retransmit-request queue for avoiding message-level deadlock is taken into account for the two deflection networks.

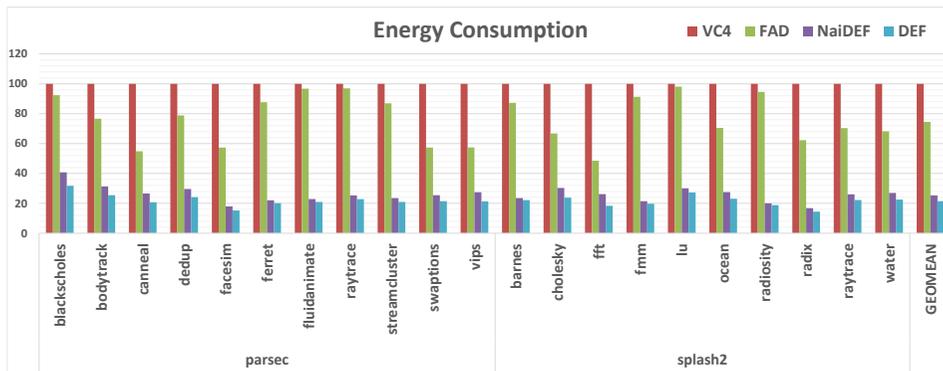


Figure 4.12: Energy consumptions on different networks.

The results shown in this section are normalized to that of buffered router with four virtual channels. Figure 4.11 shows the application latency result. Even at the overhead of retransmitting some of the messages, deflection routing performs slightly better than adaptiveXYZ in overall and significantly better than all others. In geometric mean, deflection router performs about 30.0% better than VC4, 9.2% better than naive deflection routing, and 1.7% better than adaptiveXYZ.

Figure 4.12 shows the energy consumed by the networks during the runs. In all the applications, we obtained the same order as we did in § 4.6.3 deflection being the lowest, naive deflection the second, adaptiveXYZ on third, and VC4 the worst. Again, the major source of energy difference was static/dynamic energy consumption from buffers in VC4 and adaptiveXYZ. Deflection routing had 78.6% lower energy consumption than VC4 and 71.6% lower than adaptiveXYZ. When compared with naive deflection routing, the proposed one showed 14.8% lower energy consumption.

Figure 4.13 shows the result on energy efficiency. As in Figure 4.10, the graph is drawn in log scale. Same order was obtained throughout the whole benchmark

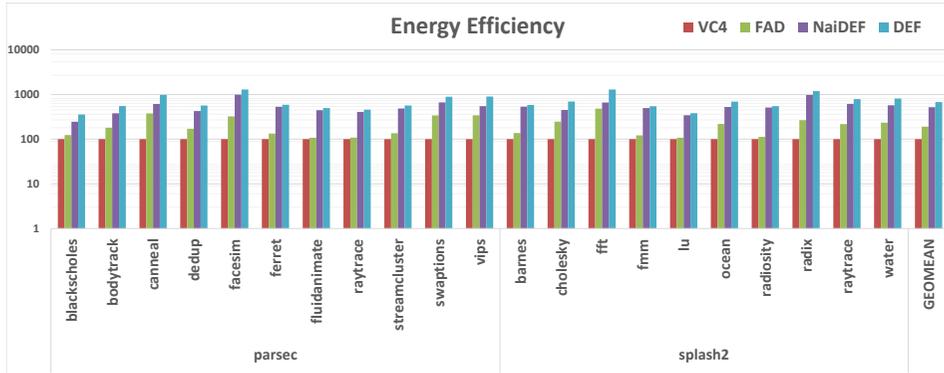


Figure 4.13: Energy efficiency on different networks.

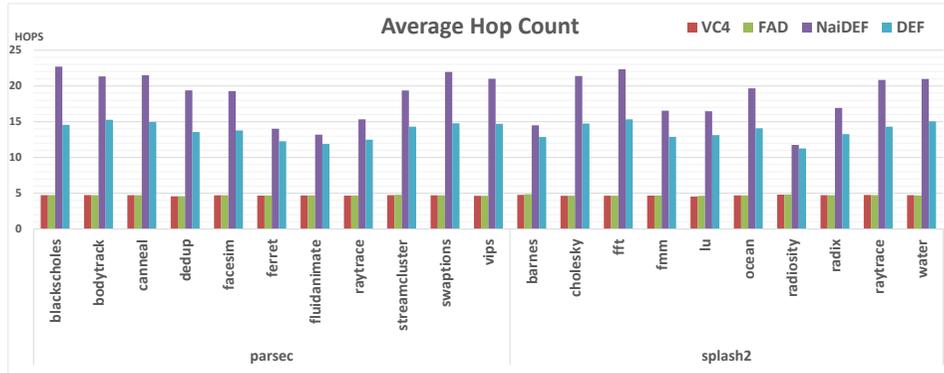


Figure 4.14: Average hops on different networks.

set. In energy efficiency, deflection routing was 6.67 times better than VC4 and 3.58 times better than adaptiveXYZ. Compared to naive deflection routing, the proposed deflection routing showed 29.2% better energy efficiency.

Figure 4.14 shows an interesting aspect: average hop counts for each network. In case of retransmission, the number of hops taken by the retransmission as well as the retransmit request has been added to the total number of hops of the packet. AdaptiveXYZ shows almost the same hop counts as VC4, since it is a minimal adaptive routing, where the hop count of a single flit transfer

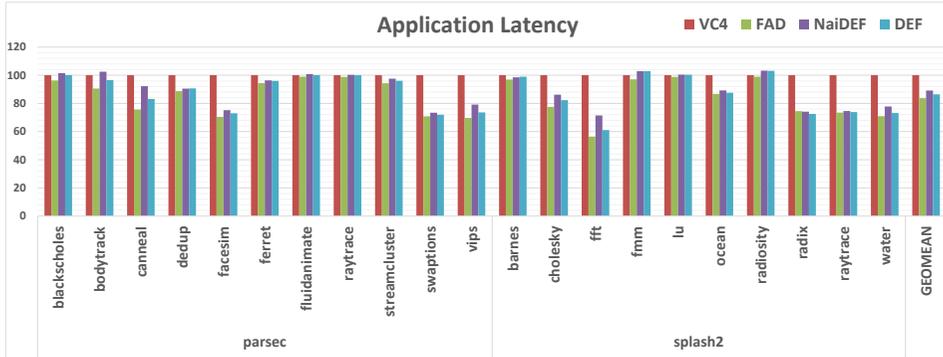


Figure 4.15: Application latencies for serialization ratio of 2:1.

is simply the minimal distance between the source and the destination. Due to deflecting and retransmitting nature, naive deflection and deflection routers show much larger hop counts; average hop count of naive deflection reaches around 20 and that of deflection is around 15. Even though the flits need to traverse three to four times longer distance, the overhead of energy consumption is more than offset by large leakage and dynamic energy consumption of buffers and virtual channel logic in buffered networks. Note that flits stay stalled in buffered routers, which explains their long latencies despite of shorter travel distance.

Although we have assumed 4:1 serialization with 1024 vertical links between layers, the number of TSVs that process or area constraint can afford may be different. For this reason, we have done experiment under serialization ratio of 2:1. Figure 4.15 through Figure 4.18 show the result. While the trends are similar to those of Figure 4.11 through Figure 4.14, the benefits have decreased by a small amount. Application latency is 13.5% lower than VC4 and 3.3% higher than adaptiveXYZ. This is because the merit of taking detour on output port conflict decreases compared to being buffered. When flits become misrouted

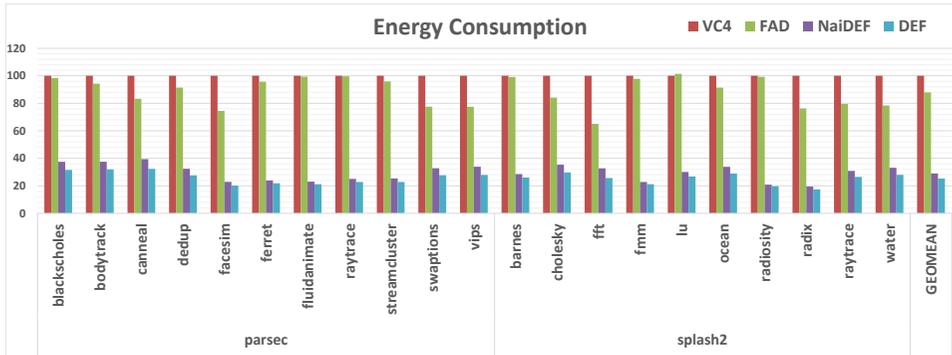


Figure 4.16: Energy consumptions for serialization ratio of 2:1.

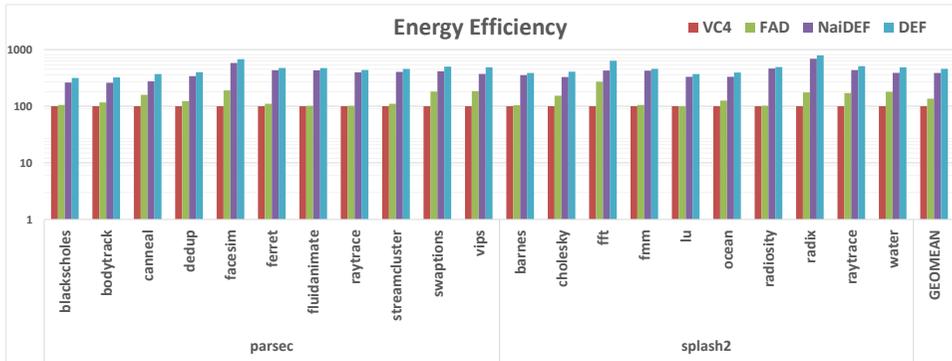


Figure 4.17: Energy efficiency for serialization ratio of 2:1.

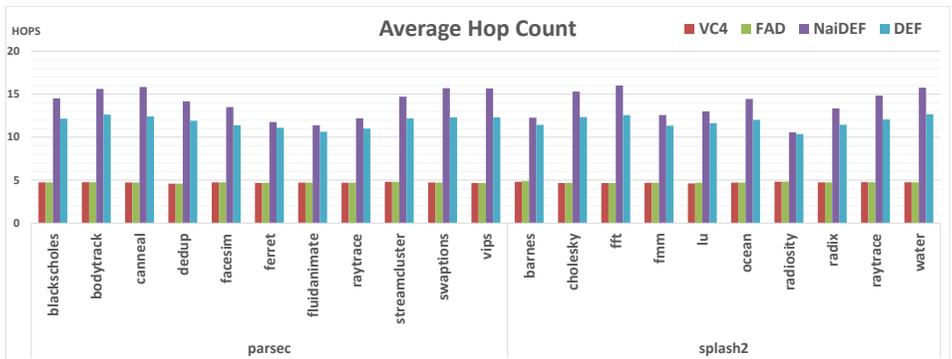


Figure 4.18: Average hops for serialization ratio of 2:1.

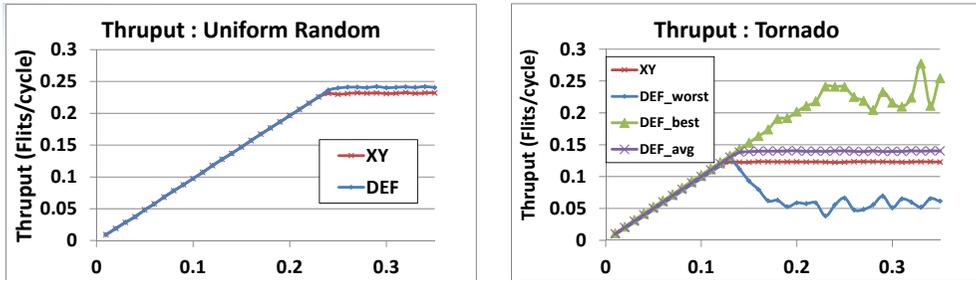


Figure 4.19: Fairness comparison with deflection routing and static routing under synthetic traffics.

due to deflection, at least 6 cycles are added to the zero-load latency because misrouting costs two hops, taking three cycles per hop. Thus, as vertical link bandwidth becomes wider, waiting for a blocked path to be cleared tends to be a better choice.

#### 4.6.5 Fairness Issue

Deflection routing relies on forwarding packets to random directions. As a result, the nodes located at the corner often suffers fairness problem. Even though the golden flit scheme guarantees the any flit eventually reaches the destination, it does not guarantee that the fairness between the nodes. Figure 4.19 shows the effect. Under uniform random traffic, there is almost no fairness issue. However, for a permutation traffic such as tornado, too much vertical traffic unveils the fairness problem. when the network is saturated, deflection routing shows around 50% worse fairness in terms of the worst received throughput. On the other hand, the deflection routing under application benchmark traffics does not show much problem. As shown in Figure 4.11, all threads with deflection routing finish earlier than those of other buffered routings. The reason is twofold: The network is not always fully saturated in practice, and the nuca traffic resembles random traffics. Although not experimented, applications

using message passing with particular patterns might experience fairness issues.

#### 4.6.6 Area Cost Comparison

One of the major benefits on deflection routing is on its area complexity. To compare area benefit of our architecture, we used DSENT [102] to estimate area cost of the routers. Table 4.3 shows area costs and their breakdown of three routers under 45nm process. In the table, “Buffers” include input buffers as well as pipeline buffers. Even though the proposed router does not have any input buffer, it has some buffer space because of pipeline buffers and a side buffer. Both buffered routers have a 7x7 crossbar. Although the proposed router does not have a traditional crossbar, the permutation logic and series of injection/ejection logic function as crossbars of traditional routers. Thus, area of those logic blocks is included in the “Crossbar” tab. For TSVs, a medium-density TSV with 20  $\mu\text{m}$  pitch was assumed and 32 links of two directions (up/down) were counted. “Control & others” includes arbitration, switch/VC allocation, and clock tree logic.

In total, the proposed router costs only 58.2% of buffered router and 26.1% of buffered router with four virtual channels. Most of the savings come from elimination of input buffers. In simple buffered router, buffers consume about 52% of total area cost and buffers in the proposed router take only 32.8% of that in the buffered router. Compared with the virtual channel buffered router, buffers of the proposed router take only 9.7% of that in the virtual channel

Table 4.3: Area Comparison and Breakdown

	Buffers	Crossbar	TSV	Control & Others	Total	Compared to Proposed
Buffered	71885	38856	25600	2474	138815	172%
Buffered, vc4	242276	38856	25600	3145	309877	384%
Deflection (Proposed)	23543	30101	25600	1488	80733	-

(Unit :  $\mu\text{m}^2$ )

router. In conclusion, our router performs comparably or better, at low energy and low area cost under a 3D environment than traditional routers.

# Chapter 5

## Routing for Partially Connected 3D NoC

### 5.1 Introduction

In this chapter, we propose a routing algorithm for a partially connected 3D mesh network. For a deadlock-free routing algorithm of 3-D NoCs with partially connected vertical links, Dubois et al. [43] have first introduced an elevator-first algorithm for an architecture that connected layers of 2-D NoCs with arbitrary partial connections of TSV links. However, considering that FIFO buffers are the second largest component of NoC power consumption [71], the dedicated VCs used in the elevator-first algorithm become the main burden for the energy efficiency (i.e., energy/performance). Also, dedicating VCs to deadlock freedom often fails in balancing loads between the VCs, and thus shows inferior performance compared to other approaches using VCs to avoid HoL (head-of-line) blockings. Our algorithm eliminates the use of virtual channel by providing a set of rules for selecting adequate elevators.

```

1 Elevator-first routing algorithm (src, current, dest)
2 //assign virtual channel
3 if (current node is the source node) {
4     if (src_z>dest_z) {
5         assign the packet to "up" vc;
6     } else if (src_z<dest_z) {
7         assign the packet to "down" vc;
8     } else {
9         assign the packet to any vc;
10    }
11 }
12 //route
13 if(current_z == dest_z ) {
14     route to dest using 2D routing algorithm (XY);
15 } else {
16     if (current node is an elevator node) {
17         route towards dest_z thru the elevator;
18     } else {
19     route to an elevator node using 2D routing algorithm (XY);
20     }
21 }

```

Figure 5.1: Pseudo-code of the elevator-first algorithm

## 5.2 Background

Our algorithm is based on the elevator-first algorithm and thus it is necessary to explain it prior to our algorithm. The elevator-first algorithm has been proposed as an algorithm that connects several layers of 2D networks that use deadlock-free algorithms. The key point of the algorithm is that it is deadlock-free even though there are only partial, arbitrary vertical connections, while only two virtual channels are needed.

A simplified pseudo code of the elevator-first algorithm is shown in Figure 5.1. When a packet is destined to a node located on another layer, it is first routed to a vertical link (an elevator) using the routing algorithm inside the 2D network (e.g., XY routing). When it arrives at the elevator, the packet is sent upward or downward (towards the destination). When the packet reaches the next layer, the destination is checked. If the destination is on the current layer, then it is routed to the destination using the routing algorithm of the

new layer. If it is not, the same process is repeated until it reaches the layer of the destination node.

The routing itself is not deadlock-free without virtual channels, but with two virtual channels only inside the 2D networks, it can be made deadlock-free. The key to ensuring deadlock-freedom is to separate the virtual channels into upward channels and downward channels. When a packet is to be routed to an upper layer, it only uses upward virtual channels and upward vertical links. When the destination is on a lower layer, it uses downward virtual channels and downward vertical links. This eliminates cycle from the channel dependency graph and deadlock-freedom is ensured. For a detailed proof, refer to [43].

It is also worth mentioning that the elevator-first algorithm requires almost no space for routing table. Assuming that each node is assigned a single fixed elevator for both (up/down) directions, it only has to store locations for the two elevators <sup>1</sup>. If each layer in the network consists of  $M$  nodes, it is only  $2\log_2(M)$  bits per router. On the other hand, for topology-agnostic algorithms which demand full routing tables, each router should store the sequence of node ids through the path for all possible destinations. In such a case, the size of the table would be  $(N - 1) \cdot D\log_2(N)$  bits per router, where  $N$  is the number of nodes in the network and  $D$  is the maximum distance of the routing path. If the maximum degree of the routers is limited, the port number can be used instead of node id. Still, the routing table size would be  $(N - 1) \cdot D\log_2(P)$  bits per router where  $P$  is the number of ports. For a 4x4x4 network, it is about 4KB, which is not a negligible size. The advantage on routing table size also holds for our algorithm.

The key difference between elevator-first and our algorithm is that elevator-

---

<sup>1</sup>We also assume that the routing within the 2D plane can be implemented with combinational logics (e.g., XY routing)

first algorithm allows any topology for the planar networks, while our algorithm narrows them to meshes. By restricting the topology, our algorithm eliminates the virtual channel usage of the elevator-first algorithm, which can lead to a significant reduction in energy consumption, or performance gain when the same number of virtual channels is used.

### 5.3 Related Work

General-purpose packet-switched 3-D NoCs, which delivers packets between processing elements, e.g., IP blocks, processors, DSP cores, memory blocks, FPGA blocks, etc., have been extensively studied for large 3-D systems on chips. In particular, there has been a large body of works that focus on extending a 2-D mesh topology to 3-D NoCs. Li et al. [118] proposed a hybridization of a bus structure and a 2-D mesh network, employing the bus structure for communications among processing elements located on different layers. Furthermore, in conjunction with hybrid bus-mesh interconnect, Kim et al. [30] proposed a dimensionally-decomposed crossbar design that makes packet contention in routers less, and Rahmani et al. [32] proposed a congestion-aware adaptive routing algorithm for inter-layer communications. Dahir et al. [119] extended the odd-even turn model for 3D meshes. Darve et al. [120] proposed a design of 3-D NoC router that has seven ports (one port to the IP block, one each to routers above and below, and one in each cardinal direction) in order to implement a true 3-D mesh (i.e., 3-D cube) network. However, these differ from our target platform in that they do not consider the limited connectivity along vertical directions.

Because of the large overhead of TSV itself, several attempts were made to reduce the number of TSVs in 3-D NoCs. Pasricha proposed serialization of TSVs for area reduction [42], and he also studied on synthesis of 3-D NoC con-

sidering TSV serialization [121]. Zhu et al. [108] suggested an adaptive routing algorithm for 3-D mesh with limited number of TSV links. Akesson et al. [122] and Liu et al. [123] proposed asymmetric mesh topologies, where only few 3-D NoC routers support inter-tier communications while the rest are allowed only horizontal communications. Such topologies can also reduce the number of vertical links, sacrificing bandwidth of inter-tier communications. In [124], the optimal number of TSVs is analyzed to reduce the chip footprint and, thus, the network latency, while sacrificing bandwidth of inter-tier communications.

For reducing power consumption of 3-D NoCs, Xu et al. [125] proposed a low-diameter 3-D interconnect network using low-radix routers. The reduced hop counts through the 3-D network consume less power. Park et al. [34] have looked at decomposing a NoC router into third dimension that helps reduce the chip footprint and power consumption. In addition, there have been researches on designing custom 3-D NoC topologies focusing on the power efficiency improvement [121, 126–128].

There have been researches on topology-agnostic deadlock-free routing for arbitrary network. Up/down routing [Schroeder et al. 1991] forms a spanning tree from the network and assign each channel up direction or down direction. Deadlock is avoided by always taking up directed channels before taking any down directed channels. Such routing can be applied to networks that allow random connections, as in wireless NoCs [Matsutani et al. 2013; Wettin et al. 2014]. Up/down routing gained popularity because of its simplicity and many variations came out for performance [Koibuchi et al. 2001; Sancho et al. 2000a; Sancho et al. 2000b], but they are not scalable because of the full lookup tables needed in each router. Borkar et al. [1988] and Wu and Sheng [1999] attempted to reduce the lookup table size. Segment-based routing [Mejia et al. 2006] divides planar meshes or tori into subnets and uses heuristics to pro-

hibit turns within each subnet and guarantee global deadlock-freedom. LASH [Skeie et al. 2002], ALASH [Lysne et al. 2006], and MROOTS [Lysne, Skeie, Reinemo and Theiss 2006] also use a deadlock-free algorithm for irregular networks, but it relies on using VCs. FDOR [Skeie et al. 2009] is another deadlock-free routing for 3-D irregular network, but it assumes the vertical links are fully connected. There have been many other attempts for simple and high performance deadlock-free algorithms on irregular networks, which were summarized in [Flich et al. 2012].

Although the aforementioned topology-agnostic routing algorithms can be used for the partially connected 3D mesh, they are far too general. Even though the vertical channels of partially connected 3D meshes can be placed randomly, the 2D network layers are planar meshes which have uniform patterns. The topology-agnostic routings do not benefit much from such uniformity. Also, they often suffer from unbalanced traffic load, because they usually utilize spanning tree to generate the algorithm. Furthermore, the irregular routing path leads to complex routing table requirements. Our algorithm greatly reduces such overhead by simply assigning elevators to each node and using existing algorithm within planar networks to reach them.

## 5.4 Proposed Algorithm

In this section, we show that, by composing some rules for choosing elevators (i.e., vertical links), elevator-first algorithm can be made deadlock-free even without VCs at all. We start with a regular and easy architecture, and then give a routing algorithm for more general and irregular architectures.

### 5.4.1 Preliminary

Before going into details of the algorithm, let us define some necessary terms, most of which are from [129] with slight modifications.

- Definition 1: An interconnection network  $I = G(V, E)$  is a strongly connected graph where  $V$  represents the set of router nodes and  $E$  represents the set of channels. Each vertex is given  $(x, y, z)$  coordinates according to their physical locations.
- Definition 2: A routing algorithm is an algorithm that implements a routing function  $R : E \times V \rightarrow E$  that maps the current channel  $e_c$  and destination node  $v_d$  to the next channel  $e_n$  so that  $R(e_c, v_d) = e_n$ .
- Definition 3:  $\text{src}(e_x)$  and  $\text{dest}(e_x)$  represent respectively the source node and the destination node of  $e_x$ .
- Definition 4: A waiting path  $P = e_0, e_1, \dots, e_k$  for routing function  $R$  is a sequence of channels where  $R(e_i, v_x) = e_{i+1}$  for some  $v_x$ . Thus, a flit in  $e_i$ , which is to be routed to  $e_{i+1}$ , may have to wait until  $e_{i+1}$  has room for it and a chain of such dependency from  $e_0$  to  $e_k$  can be constructed.
- Definition 5: A waiting cycle  $C = e_0, e_1, \dots, e_k$  is a waiting path where  $e_k = e_0$ . There should be no waiting cycle in a deadlock-free routing.
- Definition 6: A channel  $e_x$  is called an  $X+$  channel with respect to  $\text{src}(e_x)$ , if  $\text{dest}(e_x)$  has a greater x-coordinate than  $\text{src}(e_x)$ .  $X-/Y+/Y-/Z+/Z-$  channels are also defined in the same way as depicted in Figure 5.2.  $X$  channels denote both  $X+/X-$  channels and, by the same token,  $Y$  and  $Z$  channels denote  $Y+/Y-$  and  $Z+/Z-$  channels, respectively. We also call

the directions interchangeably as east ( $E$ ), west ( $W$ ), north ( $N$ ), south ( $S$ ), up ( $U$ ), and down ( $D$ ).

- Definition 7: A *turn*  $T = (e_i, e_{i+1})$  consists of two differently directed channels  $e_i$  and  $e_{i+1}$ , such that  $\text{dest}(e_i)$  equals  $\text{src}(e_{i+1})$ . Turning node is the common node of the two channels. “A path turns at node  $v$ ” means that the path has  $v$  as a turning node.
- Definition 8: An  $\alpha\beta$  *turn* ( $\alpha, \beta: E, W, N, S, U, \text{ or } D$ ) represents a turn from  $\alpha$  direction to  $\beta$  direction. For example, an  $NE$  turn represents a turn from north direction to east direction.
- Definition 9: A *row* is a set of nodes that share the same  $y$  and  $z$  coordinates. A plane or layer is a set of nodes that share the same  $z$  coordinate.
- Definition 10: A *primary direction* is one of the four planar directions ( $N, E, W, \text{ and } S$ ) that is used to order the nodes within the routing algorithm. Once the primary direction is selected (e.g.,  $S$ ), then the secondary direction is one of the the two planar directions that are not opposite to the primary direction (e.g.,  $E$  or  $W$ ). It is used to break tie when ordering nodes with the primary direction. Without loss of generality, we will use south ( $S$ ) as the primary direction and east ( $E$ ) as the secondary direction for the rest of this paper.
- Definition 11: A node  $v_x$  is called a *due – east* node of some other node  $v_y$  when  $v_x$  and  $v_y$  are on the same row and  $v_x$  is to the east of  $v_y$ . A node  $v_x$  is called a *south – or – due – east* node of some other node  $v_y$ , when  $v_x$  is either to the south or to the *due – east* of  $v_y$ . *North – or – due – west* node is defined similarly. The terms are also defined similarly for elevators, turns, and channels.

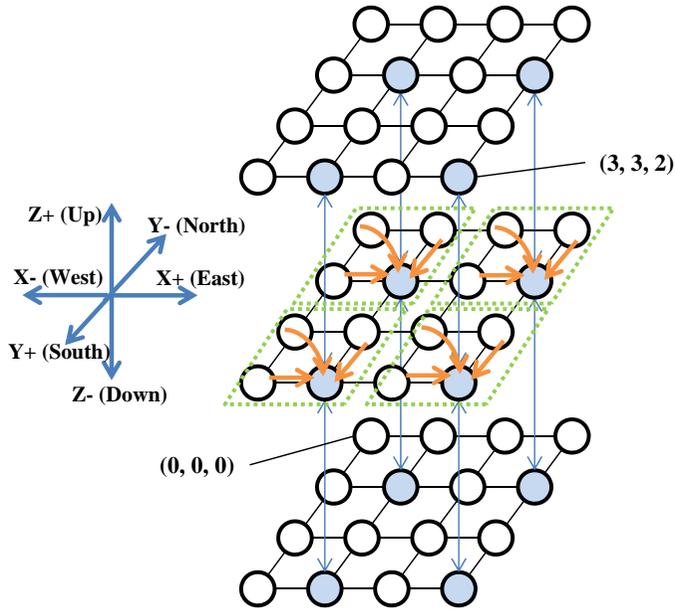


Figure 5.2: An example of 3-D stacked mesh topology with regular partial connections in the vertical direction.

- Definition 12: An up *pivot elevator* of a plane is the up elevator that has no other up elevator to the primary direction of it in the plane. If there are multiple elevators that satisfy the condition, the elevator farthest to the secondary direction is the pivot elevator. For example, if south is the primary direction and east is the secondary direction, then the up pivot elevator has no other up elevator to its south-or-due-east. The down pivot elevator is defined in the same way.

Note that an interconnection network  $I$  using routing function  $R$  is deadlock-free if no waiting path in  $I$  forms a cycle. This has been proven in [129] and we omit the proof here.

### 5.4.2 Routing Algorithm for 3-D Stacked Meshes with Regular Partial Vertical Connections

Figure 5.2 shows our first architecture to start with. In Figure 5.2, each layer is a 4x4 mesh and one vertical link is regularly placed at the south-east corner of every 2x2 block. All vertical links in the figure are bidirectional (this is assumed throughout the paper). In the architecture, deadlock-freedom of elevator-first algorithm can be obtained by one simple rule:

#### Ruleset A

- Rule A1: When the destination is on a different layer, take the elevator in the  $L \times M$  block (2x2 block in the example of Figure 5.2) that contains the source node to send the packet all the way to the destination layer.

The thick orange arrows in Figure 5.2 illustrate this rule. All packets from any of the four nodes within a 2x2 block follow  $XY$  route to the elevator in the same block. Although 2x2 blocks are used in this example, the size of a block does not actually matter and any other sized block can be used such as 3x3 or 2x1 as long as each block has an elevator at its south-east corner. Now, we claim that Rule A1 makes the routing deadlock-free, and the proof follows.

**LEMMA 1.** In a routing algorithm following Rule A1, west-up ( $WU$ ), west-down ( $WD$ ), north-up ( $NU$ ), and north-down ( $ND$ ) turns never occur.

**PROOF.** According to Rule A1,  $X$  and  $Y$  coordinates of an elevator to be taken are always equal to or greater than that of the source. If both  $X$  and  $Y$  coordinates are equal, then a  $Z$  channel of the elevator is directly taken and no turn is made. If any of the coordinates are different, by the definition of  $XY$  routing,  $E$  channel and  $S$  channel are the only channels that can be taken

before the  $U/D$  channel. Thus,  $WU$ ,  $WD$ ,  $NU$ , and  $ND$  turns never occur. (QED)

**THEOREM 1.** An elevator-first routing algorithm with Ruleset A is deadlock-free with no VC.

**PROOF.** Assume a waiting path  $P = e_0, e_1, \dots, e_n$  following elevator-first algorithm with Ruleset A.  $P$  is categorized into the following three cases.

- Case 1:  $P$  does not include any  $Z$  channel. It is guaranteed by the property of XY routing that  $P$  cannot form a cycle.
- Case 2:  $P$  includes one or more  $Z$  channels, but there is no  $Z$  directed turn (a turn into a  $Z$  direction). In this case,  $P$  starts with a  $Z$  channel but never returns to the starting point and thus cannot form a cycle.
- Case 3:  $P$  includes one or more  $Z$  directed turns ( $EU, ED, SU$ , or  $SD$ ; by Lemma 1, a turn can never be a  $WU, WD, NU$ , or  $ND$  turn). Let  $T_1$  be such a  $Z$  directed turn that is located on the north-or-due-west side of all other  $Z$  directed turns in  $P$ . In other words, no other  $Z$  directed turn in  $P$  is on the north-or-due-west side of  $T_1$ . Without loss of generality, let us assume that  $T_1$  is an upward turn. That is,  $T_1$  is either an  $EU$  turn or an  $SU$  turn. The rest of this proof shows that it is impossible for  $P$  to form a cycle.
  - If  $T_1$  is an  $EU$  turn, there should be an  $E$  directed turn just before  $T_1$  (consider the example of cycle  $UW - WD - DE - EU$ ) in  $P$  to make a cycle, and the only turns allowed into  $E$  direction in XY routing are  $UE$  turn and  $DE$  turn. Moreover, to make a cycle, there should be a  $Z$  directed turn in  $P$ , which is followed by the  $UE$  or

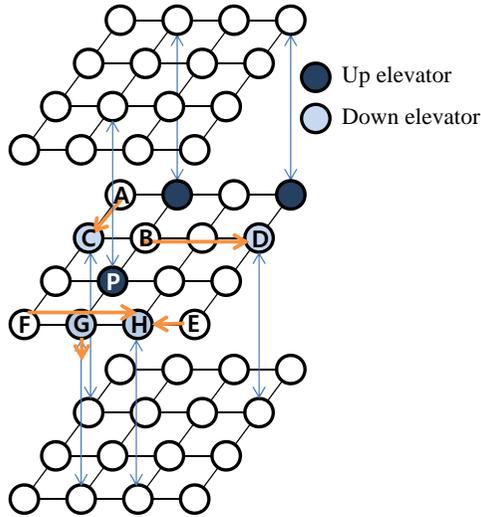


Figure 5.3: 3-D stacked meshes with random partial connections and the down elevator selection.

$DE$  turn. However, since there is no other  $Z$  directed turn on the due-west side of  $T_1$  by the assumption, no cycle can be formed.

- If  $T_1$  is an  $SU$  turn, some turn  $T_3$  has to be made into  $S$  direction in  $P$  to make a cycle. It can be one of  $US$ ,  $DS$ ,  $ES$ , or  $WS$ . Obviously,  $US$  and  $DS$  turns are impossible because there is no other  $Z$  directed turn on the north side of  $T_1$ .  $ES$  and  $WS$  turns are also impossible because only turns that can be made to  $E$  or  $W$  direction are from  $U$  or  $D$  directions, which would again violate the assumption that there is no other  $Z$  directed turn on the north side of  $T_1$ . Thus, no cycle can be formed and this proves the theorem.

(QED)

### 5.4.3 Routing Algorithm for 3-D Stacked Meshes with Irregular Partial Vertical Connections

One observation from § 5.4.2 is that a block needs not necessarily have the size of  $2 \times 2$ . Furthermore, a path does not need to take the vertical link in the same block. The only conditions used in the proof are 1) XY routing is used for planar routing and 2) four turns ( $WU$ ,  $WD$ ,  $NU$ , and  $ND$ ) are prohibited at vertical links. Thus, provided that the two conditions are satisfied, we can easily extend the algorithm to the case of irregular placement of vertical links or to the case that allows adaptiveness of selectively taking elevators according to the condition of the network. We leave the issue of utilizing the adaptiveness as future work and focus, in this section, on more irregular architectures where vertical links are randomly placed as shown in Figure 5.3

Even if we have an irregular vertical link placement, we can make the routing algorithm to take any elevator provided that the aforementioned four turns are prohibited. However, this rule poses an important restriction on the architecture; a vertical link is required at the south-east corner node of the mesh. If the node does not have a vertical link, then some nodes will have no elevator provided by the routing algorithm. In other words, the routing algorithm results in a disconnected network. Fortunately, we can solve the problem by additional rules that allow forbidden turns at certain nodes. With the addition of new rules and some adjustment to the existing rule, we can define a new ruleset for selecting an elevator.

#### Ruleset B

- Rule B1: When the destination is on a different layer, take any elevator among the south-or-due-east ones of the source (including the self-node).
- Rule B2: If there is no south-or-due-east elevator of the source in the desired

direction, take the *pivot* elevator (refer to *Definition 12* for the definition of pivot elevator).

- Rule B3: Consider a case where a down elevator is chosen by Rule B1. If the down elevator is not at the self-node but is located to the south-or-due-east of the up pivot elevator, instead of taking that down elevator, take the down pivot elevator on the same plane. The same rule applies when up direction and down direction are switched.

Note that in this ruleset, south represents the primary direction and east the secondary direction. They can be changed to other directions according to *Definition 10*. The thick arrows in Figure 5.3 show an example of elevator selection. According to Rule B1, packets from node A and B can choose down elevator C and D, respectively. However, packets from node E do not have any down elevator corresponding to Rule B1, and thus they choose the down pivot elevator H following Rule B2. The addition of Rule B3 eliminates the deadlock problem caused by adding Rule B2. Normally, packets having node “F” as their source would take node “G” as its down elevator by Rule B1. However, because “G” is located to the south of “P” (the up pivot elevator), the down pivot elevator at node “H” is selected instead according to Rule B3. On the other hand, packets having “G” as their source takes the down elevator at the self-node, “G”. We claim that this routing algorithm is deadlock-free. Please note that the pivot elevator is one of the existing elevators, not a newly added one.

**LEMMA 2.** In a waiting path  $P = e_0, e_1, \dots, e_n$  constructed following Ruleset B, let  $e_i$  ( $0 \leq i \leq n$ ) be an up segment and  $e_j$  ( $0 \leq j \leq n$ ) be a down segment in  $P$  where  $i < j$  and  $\text{src}(e_i)$  and  $\text{dest}(e_j)$  be on the same plane. Then  $\text{dest}(e_j)$  is to the south-or-due-east of  $\text{src}(e_i)$ . The same holds when  $e_i$  is a down segment

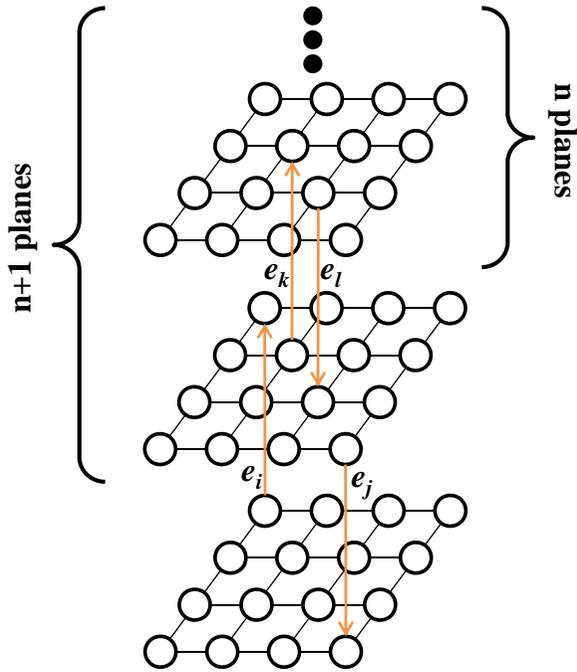


Figure 5.4: Illustration of the proof of Lemma 2.

and  $e_j$  is an up segment.

**PROOF.** We prove it by induction. Without loss of generality, let  $e_i$  be an up segment. We will first show that Lemma 2 is true when there is only one plane above  $\text{src}(e_i)$ , and then show that if it is true when there are  $n$  planes above it, it is also true when there are  $n + 1$  planes above it. The illustration of the proof is shown in Figure 5.4. If there is only one plane above  $\text{src}(e_i)$ , by Rule B1, the down elevator that can be taken must be on the south-or-due-east side of  $\text{dest}(e_i)$  (the only exception is Rule B2, but an elevator on the north-or-due-west side of  $\text{dest}(e_i)$  cannot be a pivot elevator and thus cannot be taken). Therefore,  $e_j$  (and thus  $\text{dest}(e_j)$ ) must be on the south-or-due-east side of  $e_i$  (and thus  $\text{src}(e_i)$ ). Now, let's assume that the lemma is true when there are  $n$  planes upward. Assume first that the path between  $e_i$  and  $e_j$  never reaches the

plane upper than the plane of  $\text{dest}(e_i)$ . Then  $e_j$  is to the south or due east of  $e_i$  just as the case of only one upper plane existing. Now, assume that the path has an upward channel  $e_k$  ( $i < k < j$ ) after  $e_i$  and  $\text{dest}(e_i)$  and  $\text{src}(e_k)$  are on the same plane. Then there can be two cases:

- Case 1:  $e_k$  is on the south-or-due-east side of  $e_i$  or at the same  $(x, y)$  location as  $e_i$ . Assume that there are  $n$  planes above  $\text{src}(e_k)$ , down channel  $e_j$  ( $k < l < j$ ) in  $P$  has destination node on the same plane as  $\text{src}(e_k)$ , and  $e_j$  is on the south-or-due-east side of  $e_k$ . Then  $e_j$  must be on the south-or-due-east side of  $e_i$ . Otherwise,  $e_j$  should have been a down pivot elevator since it should have been taken by Rule B2 after  $e_j$  even if it were located on the west-or-due-north side of  $e_j$  (recall the relative location of  $e_i$  and  $e_j$ ). In this case, there would have been violation of Rule B3 since  $e_k$  is not an up pivot elevator (up elevator at the location of  $e_j$  is actually the up pivot elevator).
- Case 2:  $e_k$  is a north-or-due-west channel of  $e_i$ : This means  $e_k$  is the up pivot elevator. Assume that down channel  $e_j$  ( $k < l < j$ ) in  $P$  has destination node on the same plane as  $\text{src}(e_k)$ , and  $e_j$  is a south-or-due-east channel of  $e_k$ . This is contradiction to the fact that  $e_k$  is an up pivot elevator (up elevator at the location of  $e_j$  is the up pivot elevator). Thus, this case never occurs.

Therefore, by induction, we can say that the lemma always holds. (QED)

**THEOREM 2.** The elevator-first routing algorithm with Ruleset B is deadlock free with no VC.

**PROOF.** Let  $P = e_0, e_1, \dots, e_n$  be a waiting path constructed following Ruleset B. If there is no  $Z$  channel in  $P$ , no cycle can be formed by the property

of XY routing. Now assume that there are one or more  $Z$  channels in  $P$  and let the first  $Z$  directed turn in  $P$  be  $T_1 = (e_i, e_{i+1})$  and its turning node be  $t_1$ . By Lemma 2, any other  $Z$  channel in  $P$  that has the destination node on the same plane as  $t_1$  is on the south-or-due-east side of  $t_1$ . If  $T_1$  is one of  $NU, ND, WU$  or  $WD$  turns, it means that  $e_{i+1}$  is the pivot elevator. However, by Lemma 2 again, any  $Z$  segment in  $P$  having the destination on the same plane as  $t_1$  must be on the south-or-due-east side of  $t_1$  (or  $e_{i+1}$ ), implying that  $e_{i+1}$  cannot be a pivot elevator, which is a contradiction. Thus this case never happens. If  $T_1$  is one of  $EU, ED, SU$  or  $SD$  turns,  $T_1$  cannot be reached from south-or-due-east nodes by XY routing algorithm. Therefore,  $P$  cannot form a cycle, and the algorithm is deadlock-free. (QED)

#### 5.4.4 Extension to Heterogeneous Mesh Layers

In 3-D stacking technology, each layer can be used for different purposes and thus different technology be used. For example, processor cores can be placed on one layer and memory can be placed on another layer [122]. Consequently, maintaining same network topology in each layer can be difficult in some situations. In this section, we show that the same routing algorithm can be used even for heterogeneously sized mesh layers while maintaining deadlock-freedom, if the topology preserves the “relative locations” between elevators. Informally, preservation of relative locations can be explained as follows. An elevator is said to connect a source router and a destination router at source layer and destination layer, respectively. In a topology where the relative locations of elevators are preserved, if the source routers of two elevators have the same x-coordinate, the destination routers of the corresponding elevators should also have the same x-coordinate. Similarly, if the source router of an elevator has a larger x-coordinate than the other source router, the destination router of

that elevator should also have a larger x-coordinate. Of course, the same holds for the y-coordinates. Under this condition, all characteristics that we used to prove Ruleset B hold for the heterogeneously sized meshes. Considering that tilted TSVs are usually not implemented for any reason, preserving the relative locations do not pose a restriction to the topology. To make a formal definition, we first define two sets  $SX_n$  and  $SX_n$  as follows:

- $SX_n = \{sx_{ni} \mid v_i \text{ is a node in layer } n, v_i \text{ is connected by a vertical link, and } sx_{ni} \text{ is the x-coordinate of } v_i\}$
- $SX_n = \{sy_{ni} \mid v_i \text{ is a node in layer } n, v_i \text{ is connected by a vertical link, and } sy_{ni} \text{ is the y-coordinate of } v_i\}$

and two mappings  $f$  and  $g$  as follows:

- $f : SX_n \rightarrow SX_m$  is a mapping between x-coordinates in layer  $n$  and those in layer  $m$  implemented by vertical links. Thus,  $sx_{mj} = f(sx_{ni})$  represents a vertical link from a node with x-coordinate  $sx_{ni}$  to a node with x-coordinate  $sx_{mj}$ .
- $g : SX_n \rightarrow SX_m$  is a mapping between y-coordinates in layer  $n$  and those in layer  $m$  implemented by vertical links.

Thus if there is a vertical channel  $e_i = (v_i, v_{i+1})$  and  $v_i$  is located at  $(x, y)$ , then  $v_{i+1}$  is located at  $(f(x), g(y))$ . The mappings  $f : SX_n \rightarrow SX_m$  and  $g : SX_n \rightarrow SX_m$  must be monotone increasing functions for any layer  $n$  and  $m$ . That is,

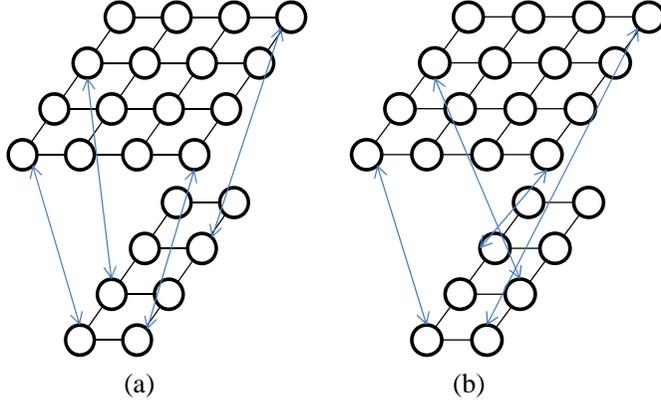


Figure 5.5: Examples of heterogeneous mesh architecture. (a) An architecture that satisfies the topological conditions and (b) the other one that does not.

the mappings must satisfy the following topological conditions.

$$sx_{n1} > sx_{n2} \leftrightarrow f(sx_{n1}) > f(sx_{n2})$$

$$sx_{n1} = sx_{n2} \leftrightarrow f(sx_{n1}) = f(sx_{n2})$$

$$sx_{n1} < sx_{n2} \leftrightarrow f(sx_{n1}) < f(sx_{n2})$$

$$sy_{n1} > sy_{n2} \leftrightarrow g(sy_{n1}) > g(sy_{n2})$$

$$sy_{n1} = sy_{n2} \leftrightarrow g(sy_{n1}) = g(sy_{n2})$$

$$sy_{n1} < sy_{n2} \leftrightarrow g(sy_{n1}) < g(sy_{n2})$$

Figure 5.5 shows two examples: (a) that satisfies the topological conditions and (b) that does not. This set of restrictions maintains relative locations of nodes connected by vertical links in a 3-D structure of meshes of different size and shape. Because all TSVs are actually manufactured in perpendicular to the dies, the conditions are practically not restrictions.

**THEOREM 3.** The routing algorithm described in § 5.4.3 on heterogeneously-

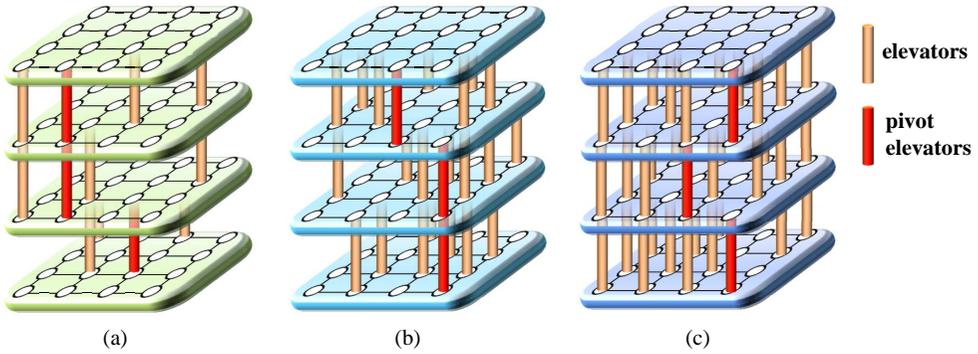


Figure 5.6: Partially connected mesh architectures for experiments. (a), (b), and (c) have 25%, 50%, and 75% of vertical connections.

sized mesh architecture that satisfies the topological conditions is deadlock-free with no VC.

**PROOF.** The proof is done by re-numbering the coordinates. Let us choose the top-most layer  $l$  as a reference layer. For the adjacent layer  $m$ , there exist functions  $f$  and  $g$  such that  $sx_{mj} = f(sx_{li})$  and  $sy_{mj} = f(sy_{li})$ . Then, re-number all coordinates of the connected nodes in  $m$  and  $l$  such that  $sx_{mj} = sx_{li}$  and  $sy_{mj} = sy_{li}$ . The re-numbering is done by assigning the bigger value to each pair (e.g., if  $sx_{mj}$  is bigger, then the value is assigned to  $sx_{li}$ ). Give all other nodes proper coordinate values relative to the nodes already re-numbered (this process is similar to image warping). In this way, all the layers can be re-numbered. Then, all the  $Z$  directed channels have their source and destination nodes with matching  $(x, y)$  coordinates. Then all the architectural assumptions used for Theorem 2 still hold, and thus Theorem 3 can be proven in the same way. (QED)

## 5.5 Experimental Results

### 5.5.1 Experimental Setup

We performed experiments using 3-D NoC architectures consisting of four layers of 4x4 mesh topology as shown in Figure 5.6. The core clock frequency and NoC clock frequency are, respectively, assumed to be 2.6GHz and 2.0GHz. The router was 3-stage pipelined. Each port has 8-flit buffers and the buffer-bypass technique was used. To evaluate 3-D NoCs under various conditions of partial vertical connections, we randomly generated three partial connected 3-D mesh architectures which have 25%, 50%, and 75% of vertical connections. To examine the NoC performance, an in-house, cycle-accurate NoC simulator was used. For energy comparison, DSENT [102], an NoC power modeling tool, was used to model the components of NoC routers and wires. For TSVs, model from [112] was used. Table 5.1 shows the parameters used for the energy model in this paper.

For the benchmarks, synthetic traffic patterns as well as traces from real applications were used. Four kinds of synthetic traffic patterns were used: uniform random, hotspot random, bit complementary, and tornado. Under the traffics, three metrics including average latency, aggregate throughput, and en-

Table 5.1: Network Design Parameters

Resource	Value
Technology Node	45nm
Network Size	4x4x4
Frequency	2.0 GHz
TSV Length	100 $\mu$ m
Wire Length	1 mm
Router Pipeline Latency	2 cycles
Wire Latency	1 cycle
Buffers per Virtual Channel	8 flits
Channel Width	128 flits

ergy consumption per flit were measured. In the simulation, the networks were warmed up for 100,000 cycles and run for another 100,000 cycles. After that, it was run without injecting any more packets into the network until all packets reached the destination to ensure there was no deadlock. For real applications, traces of one billion instructions from each of 21 multi-threaded applications in SPLASH-2 [114] and PARSEC [115] were obtained using Sniper multi-core simulator [116]. For the evaluation of real applications, we assumed that each router is connected to a node that has an x86 processor, a private L1 cache, and a slice of shared L2 cache. The L2 cache is configured as a low-bit interleaved S-NUCA [117] and assumed to be perfect. We modeled the coherent cache system using MESI protocol. Traffics from cache misses as well as coherence messages were modeled and used to stress the 3-D NoC network.

For the evaluation of the proposed algorithm, we consider the three routing algorithm with the corresponding NoC architecture as follows.

- **Elf**: the baseline elevator-first algorithm proposed in [43]. It uses two VCs for the planar links.
- **Redelf**: the proposed algorithm in this paper with no VCs.
- **Redelfv2**: the proposed algorithm using two planar VCs. Since the baseline elevator-first algorithm (i.e., Elf) uses two VCs for the planar links, comparing it with the proposed algorithm (i.e., Redelf) wouldn't be fair in the view of performance. Obviously, the network with more VCs would be better in the latency and throughput, while suffering from power and area overhead of larger buffers and VC allocators. Thus, we also tested the proposed algorithm in this paper with two planar VCs in order to compare it with the baseline elevator-first algorithm. However, instead of

using the VCs to avoid deadlock, we used them to mitigate head-of-line blocking.

Unlike synthetic traffic patterns, the real application traffics require more VCs to avoid protocol deadlocks in addition to the ones needed for routing deadlock avoidance. The coherence protocol we use requires two VCs, one for request messages and the other for reply messages. Thus for the real application benchmarks, the usage of virtual channels are doubled, which gives two for “Redelf” and four for “Elf” and “Redelfv2”.

For real application workloads, we compare our algorithm with an additional routing algorithm, Mroots, designed for arbitrary topology.

- **Mroots:** The problem of up\*/down\* routing is its traffic congestion around the root. Mroots uses virtual channels given for protocol deadlock avoidance to distribute traffic across multiple roots. Within each virtual channel, packets are routed using up\*/down\* routing with different spanning trees [130]. The roots of the trees are placed as far as possible to avoid overlaying of congested regions. In our experiment, each message type required for the coherence protocol (i.e., request and reply) has its own dedicated independent virtual channel. Therefore, Mroots uses two virtual channels in total, which is the same as ”Redelf”.

### 5.5.2 Experiments on Synthetic Traffics

Figure 5.7 shows the latency results of the routing algorithms (i.e., Elf, Redelf, and Redelfv2) under different traffic patterns (i.e., uniform random, hotspot random, bit complementary, and tornado) on a 3-D NoC with different number of vertical links (i.e., 25%, 50%, and 75%). As shown in Figure 5.7, “Redelfv2” shows the best latency in all the cases. This comes mainly from a better utilization of the VCs. Even though “Elf” also uses two VCs, the VCs can be

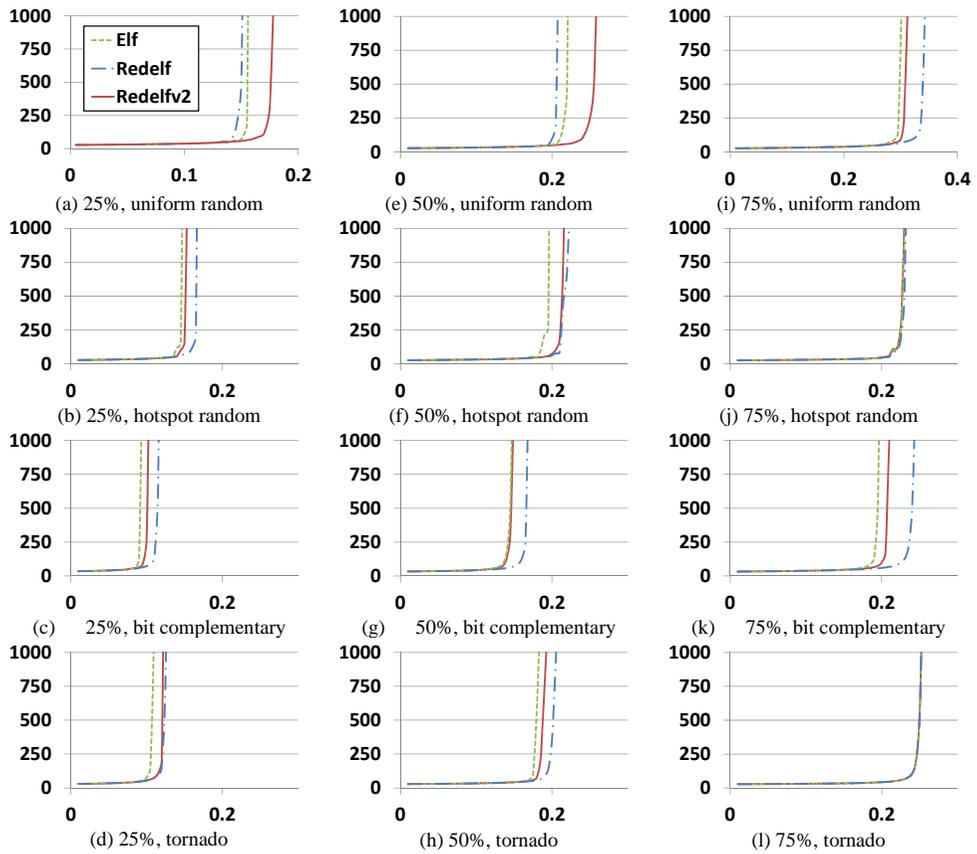


Figure 5.7: Comparison on average latencies over various number of vertical connections and traffic patterns.

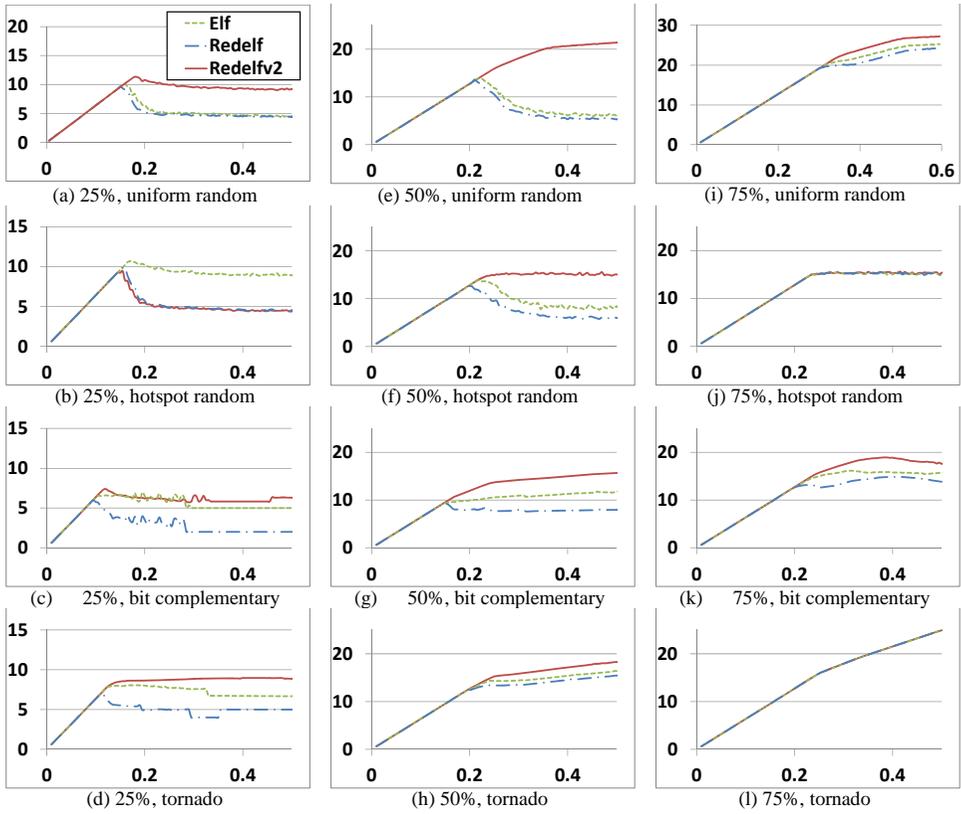


Figure 5.8: Comparison on aggregate throughputs over various number of vertical connections and traffic patterns.

unbalanced because the direction of the packets decides which VC the packet would take. “Redelf” shows the longest latency, which certainly comes from the lack of VCs. However, the difference between “Redelf” and “Elf” is relatively small. To make a quantitative comparison, we can define “saturated point” as the point where the latency reaches over 500 cycles. On average, saturation point of “Redelfv2” is 8.4% higher than “Elf”, and that of “Redelf” is 4.5% lower than “Elf”.

Figure 5.8 shows the aggregate throughputs of each routing algorithm. In Figure 5.8, “Redelfv2” clearly shows the best results. One important observa-

tion is that in many cases, “Elf” and “Redelf” show severe throughput drop after they reach the top performance as injection ratio increases, which comes from congestion at the vertical channels when there are small number of vertical channels. This is rarely observed in “Redelfv2” and the effect is not severe. Because overshooting is observed in the graphs, we compare the throughput in terms of peak throughput and saturated throughput. In average peak throughput, “Redelfv2” is 13.7% higher than “Elf”, and “Redelf” is only 6.0% lower than “Elf”. In terms of average saturated throughput, “Redelfv2” is 31.2% higher than “Elf”, and “Redelf” is 10.6% lower than “Elf”. In summary, our algorithm is superior to the baseline algorithm when the same amount of resources (i.e., VCs) is used, and even when used without VC, the performance drop is relatively small.

Figure 5.9 shows the results of energy consumption per flit (EPF) for each routing algorithm. At the low injection ratio, the EPF results are very high because the static energy of the network dominates the total energy consumption. As injection ratio increases, the EPF results drop because the dynamic energy dominates. Under the conditions where the throughput drops, the EPF results also drop along. Mainly because of throughput differences, “Redelfv2” tends to show lower EPF than “Elf”. “Redelf” usually consumes the lowest EPF, especially under low injection ratio. This comes from the smaller hardware overhead. However, under some conditions such as the one with uniform random traffic on 50% vertical channel architecture, EPF starts to rise after some point because the throughput drops and so does the energy efficiency. In terms of lowest EPF, “Redelf” consumes 27.9% less energy than “Elf” on average, and “Redelfv2” consumes 8.0% less than “Elf”. For saturated EPF, “Redelf” is 17.7% lower than “Elf” and “Redelfv2” is 27.7% lower than “Elf”. This implies that, when designing for low-power, “Redelf” has advantage in

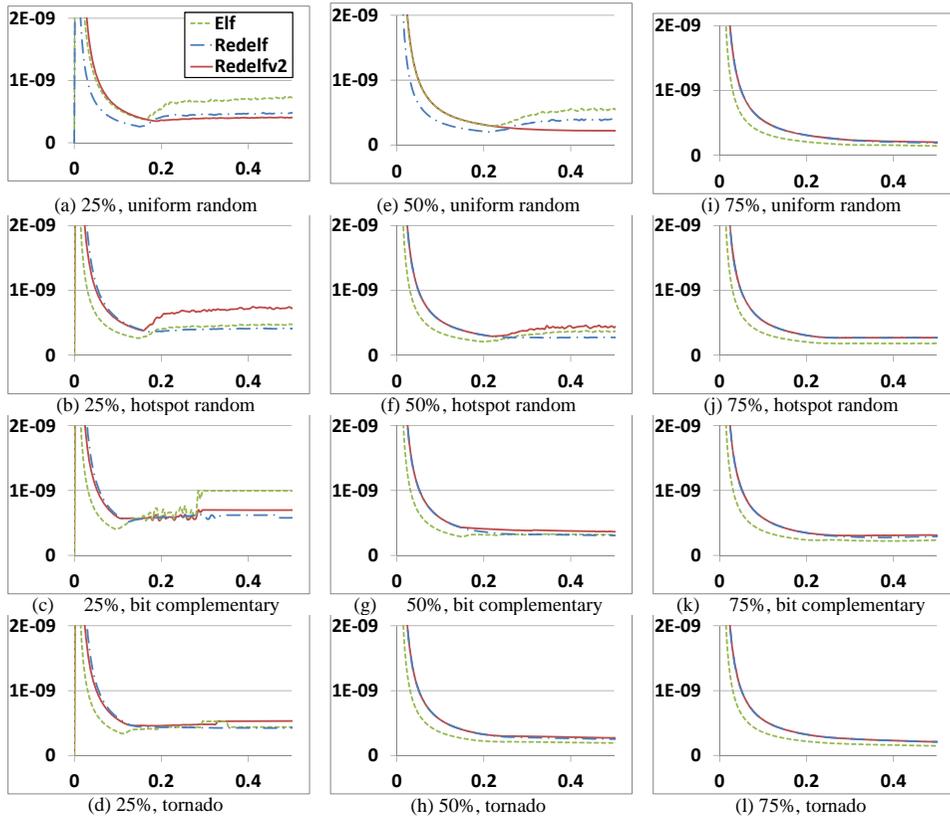


Figure 5.9: Comparison of EPF over various number of vertical connections and traffic patterns.

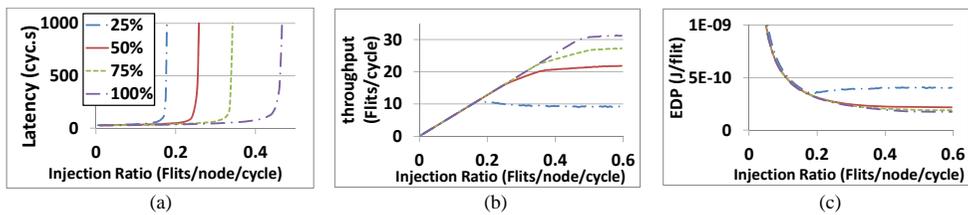


Figure 5.10: Comparison of average latency, aggregate throughput, and EPF over various number of vertical connections under uniform random traffic pattern.

energy efficiency when the network is expected to work at relatively low load, and “Redelfv2” should be better when there is high stress in the network.

Figure 5.10 shows how the reduced number of vertical links directly affects system performance of the network. Because small number of vertical links limits the bisection bandwidth, the throughput/latency is roughly proportional/inversely proportional to the number of vertical links. EPF, on the other hand, shows less difference depending on the number of vertical links. This is because the energy efficiency gained from the increased throughput is offset by the increased power consumption due to more links, buffers, and logic components.

### 5.5.3 Experiments on Application Benchmarks

Figure 5.11 shows the application latency (execution time) results of the routing algorithms (i.e., Elf, Mroots, Redelf, and Redelfv2) on a 3-D NoC with different number of vertical links (i.e., 25%, 50%, and 75%) for real applications. As expected, “Redelfv2” shows the best latency in all the cases, up to 23.1% (on average 6.9%) reduction in latency compared to “Elf”, because of the better utilization of the VCs. Although “Redelfv2” uses the same number of VCs as “Elf”, it has more freedom to assign packets to VCs while “Elf” has fixed assignment to avoid routing-induced deadlocks as well as protocol-induced deadlocks. Compared to “Elf”, “Redelf” shows only less than 10.0% (on average 1.7%) increase in latency. “Mroots” shows the worst performance, because its spanning tree based routing incurs too much traffic congestion around the two roots. Furthermore, their two virtual channels are allocated to the request and reply message types, one for each, and they do not try to balance the traffic among the virtual channels. “Mroots” shows 42.9% longer latency on average. Table 5.2 shows some hints on the latency results. It shows the average zero-load latencies

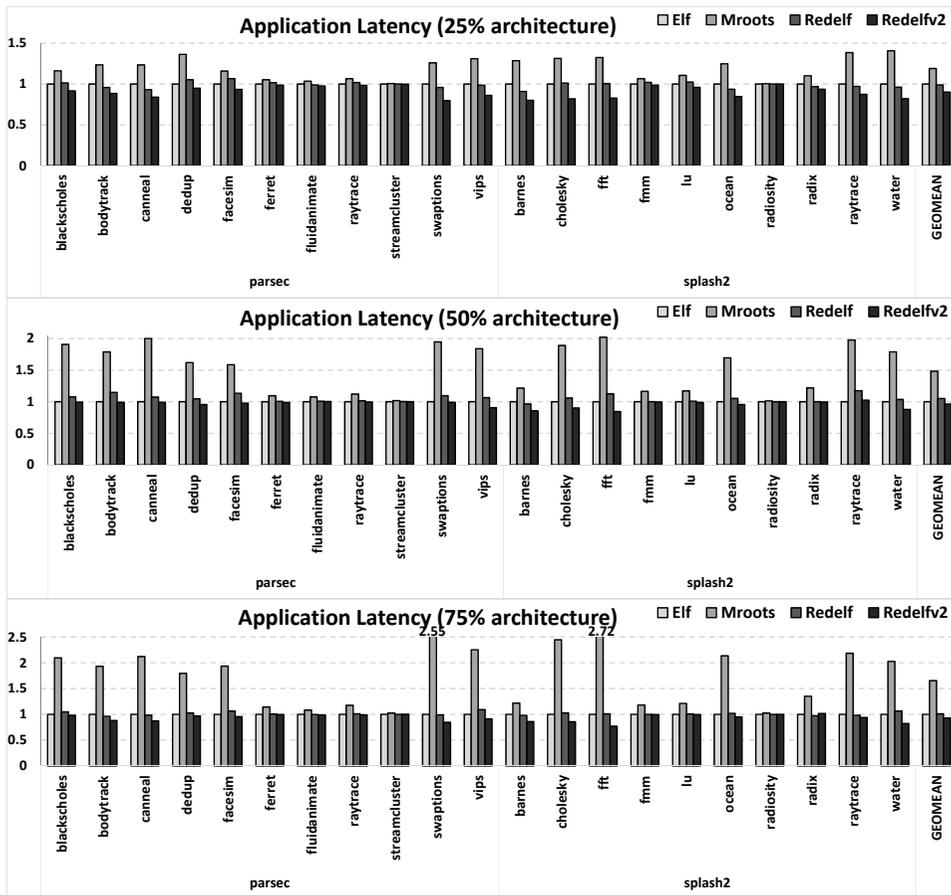


Figure 5.11: Application latencies on different routing algorithms.

of the routing algorithms under uniform random traffic, and the ideal latency obtained by the shortest-path algorithm. Because “Redelf” has some restriction on its elevator selection, the zero-load latency of “Redelf” is longer than that of “Elf”. This, together with the lack of virtual channel, makes “Redelf” slower than “Elf”. However, “Redelfv2” beats “Elf” despite the longer zero-load latency because it has more balanced traffic across the virtual channels. Mroots, however, shows longest application latency since it has long zero-load latency and still suffers from the near-root congestion even with the two roots. One more thing to note is that the difference between ideal zero-load latency and latencies of others is smaller on the 75% mesh topology, because the effect of restriction on selecting elevators becomes weak as the number of elevators increases.

Figure 5.12 shows the result of energy consumption. “Redelf” yields up to 44.6% (on average 38.9%) reduction in energy consumption compared to “Elf”, mainly due to the reduction in the VCs. The buffers in VCs take a very large portion of the energy consumption in NoC routers. Even though the running time increases, power reduction coming from the elimination of VC allocation logic together with less number of buffers does more than offsets the excess running time, resulting in the total energy reduction. Thus “Redelf” gains significant energy efficiency. Compared to “Elf”, “Redelfv2” yields up to 20.1% (on average 6.4%) reduction in energy consumption, even though it uses

Table 5.2: Zero-load Latencies(Cycle)

Routing	Topology		
	25%	50%	75%
Elf	21.9	20.5	19.7
Redelf	22.4	21.1	20.1
Mroots	29.1	25.5	25.4
Ideal	19.6	18.2	17.6

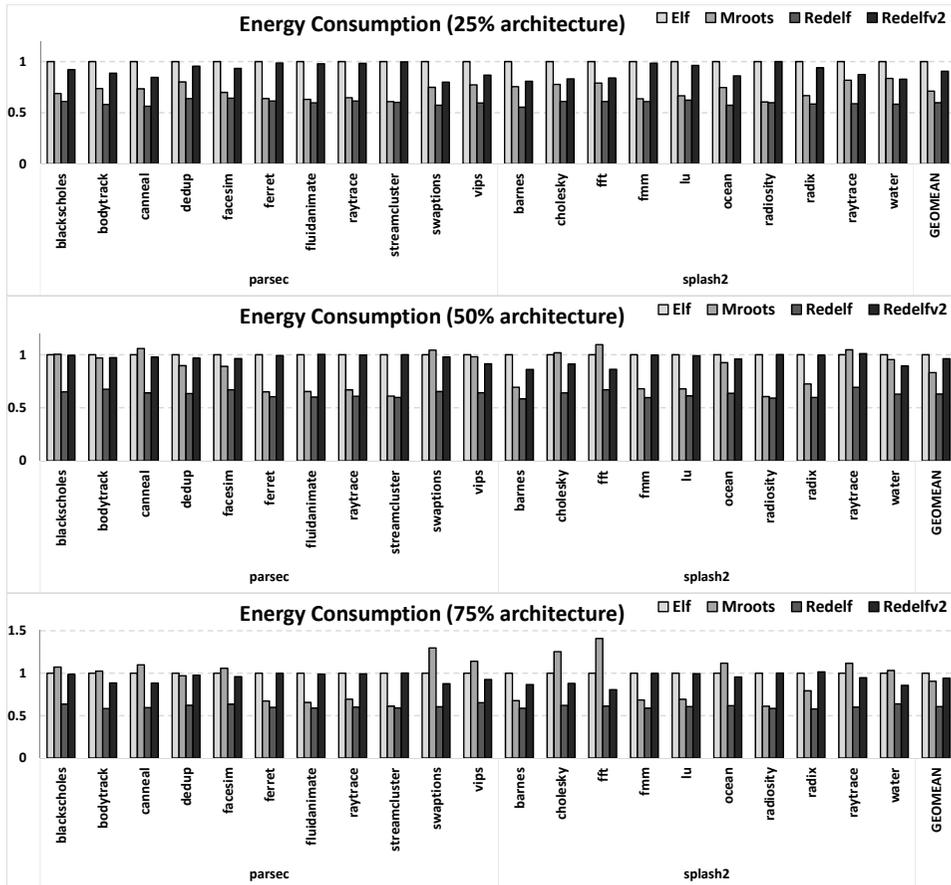


Figure 5.12: Energy consumptions on different routing algorithms.

the same number of VCs as “Elf”. It is mainly due to the reduced running time. “Mroots” has low static power and dynamic energy per access, because it uses relatively small amount of hardware resources like “Redelf”. However, the reduction due to small hardware is mostly offset by the long running time. As a result, the average energy saving of “Mroots” over “Elf” is only 18.8%, which is 20.1 percent point less than that of “Redelf”.

Considering that there is always a trade-off between energy and latency, we use EDP (Energy-Delay Product) as a metric to represent energy efficiency. The EDP over the baseline is shown in Figure 5.13. “Redelf” shows up to 49.6% EDP reduction and 37.9% reduction on average from the baseline “Elf”. The reduction mainly comes from the energy consumption of less virtual channels and buffers. The marginal performance drop does not affect much of the EDP metric. “Redelfv2”, on the other hand, obtains reduction from both execution time and energy consumption. As a result, “Redelfv2” shows up to 36.3% reduction and 12.8% on average compared to “Elf”. In case of “Mroots”, large performance loss and small energy reduction result in 16.0% increase in EDP over the baseline “Elf”. Advantage in EDP does not always mean superiority because reducing latency is sometimes much harder than reducing energy or vice versa. However, the significant difference in EDP implies that our design uses energy far more efficiently.

It is also worth noting that the benefits from “Redelf” get slightly worse as the number of elevators increases. It is because, as the number of elevators increases, the 3-D NoC becomes similar to ordinary meshes (i.e., 3-D cube), and congestions at the elevators become less severe. In result, the latency reduction becomes smaller as well as the reduction in the static energy consumption. However, the trend remains the same throughout all the 3-D NoC architectures.

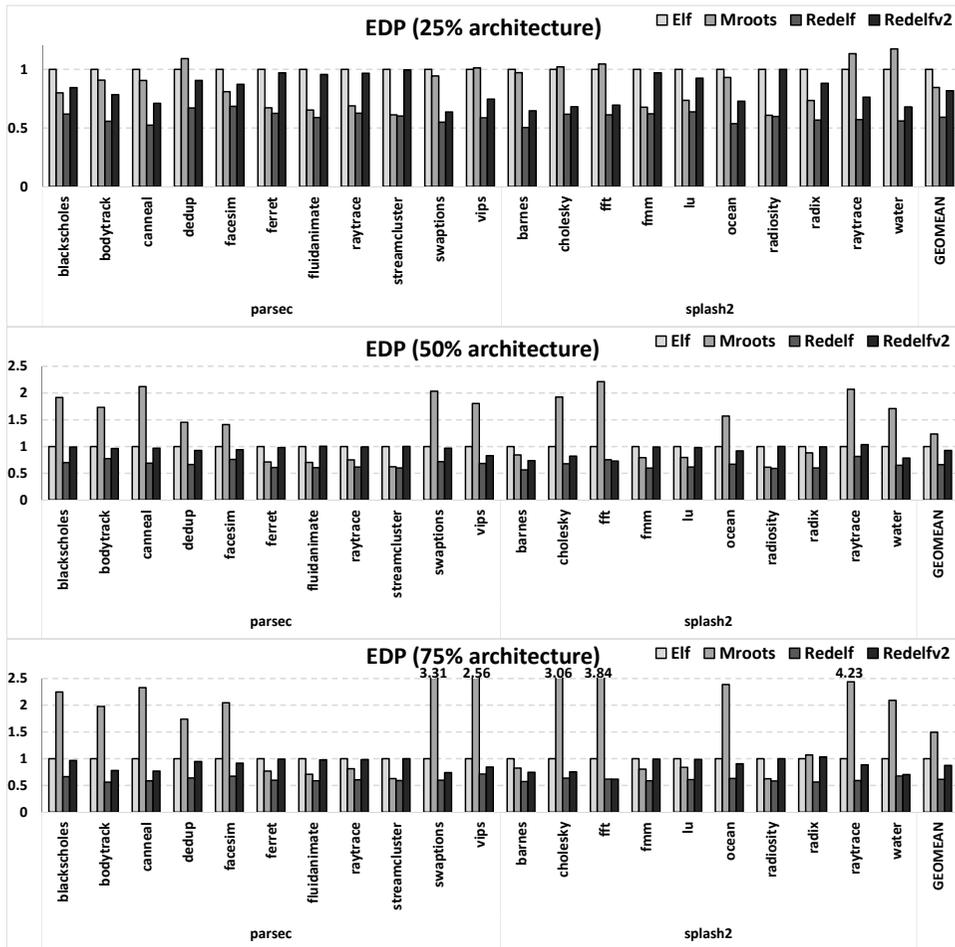


Figure 5.13: Energy-delay products of different routing algorithms.

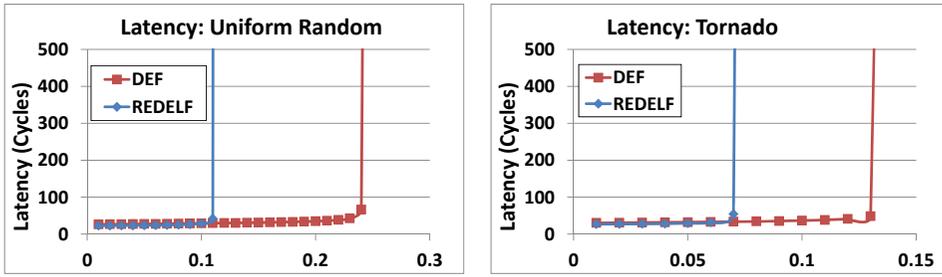


Figure 5.14: Comparison of partially connected mesh and reduced bandwidth mesh under synthetic traffics.

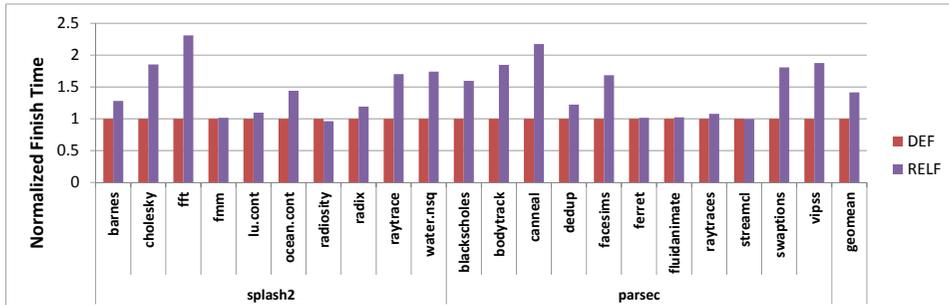


Figure 5.15: Comparison of partially connected mesh and reduced bandwidth mesh under application traffics.

### 5.5.4 Comparison with Reduced Bandwidth Mesh

When the number of vertical links are limited, there are two options, as introduced in §4 and in this chapter. Figure 5.14 shows the comparison of latency under uniform random and tornado traffics. Under the two traffics, reduced bandwidth mesh shows superior performance over the partially connected mesh. The routers in a reduced bandwidth have higher degrees on average, and thus provides more crossbar throughput. In contrast, all the packets destined to other layers gather towards a few elevators. As a result, severe congestion occurs at the vertical ports, resulting in poor performance. Applications benchmarks show consistent results, and the reduced mesh performs 41.5% better as shown

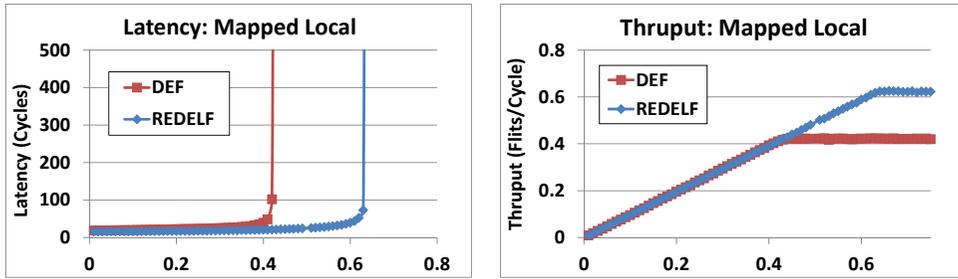


Figure 5.16: Comparison of partially connected mesh and reduced bandwidth mesh under locally mapped application.

in Figure 5.15.

On the other hand, when we assume that the running applications is mapped considering the elevator locations or the elevators are placed considering the application traffics, partially connected meshes can perform better then the reduced bandwidth meshes. To simulate such cases, we designed a mapped random traffic. In the traffic, non-elevator nodes send packets only to the nodes in the same layer. When a node is connected to an elevator, on third of the packets are sent to the adjacent layer. Figure 5.16 shows the result. in the mapped traffic, the partially connected mesh shows better performance in both terms of latency and bandwidth. In conclusion, reduced bandwidth with deflection routing is better in general, but if application characteristics and their traffic are known a priori, partially connected meshes can be a good design choice.

# Chapter 6

## Conclusion

In this thesis, we introduced four techniques for NoC in various design levels that can improve the programmability, performance, and energy efficiency. The first technique considers the communication type, when mapping and scheduling tasks on a NoC-based many-core SoC. The second technique performs thermal management of NoC with cores on 3D stacked multi-core processors. The third technique applies deflection routing on 3D NoC for efficient packet transfer, and the fourth technique proposes a deadlock-free routing for partially connected 3D meshes.

In the first part, we developed an automatic mapping and scheduling algorithm based on probabilistic algorithms and modulo scheduling, which also considers mapping of communication types to message passing or shared memory. The approach gives better results by exploring larger design space compared to the approaches considering a single type of communication. The experimental results show that we can get better energy consumption, performance, and energy-delay product depending on the objective function we choose. From the

programmer’s point of view, he/she needs to provide only parallelized application graphs with their associated values (execution time and communication volume). This is not different from typical application parallelizing processes. With the proposed approach (integrated into a parallelizing compiler), however, the programmer does not need to consider communication types during the software development since the decisions are made automatically, which can significantly improve the developer’s productivity.

In the second part, we proposed THOR, a framework for thermal management of 3D CMPs that manages network routers as well as cores in a harmonized manner. It takes the characteristics of applications into account to have efficient thermal control of cores and routers and thus maximizes the system-wide performance under the temperature constraint. Over the baseline, the proposed framework improves the performance by 18.1%. Our future work includes deriving a performance/power estimation model for out-of-order cores.

In the third part, we showed a kind of deflection routing that can be used in 3D NoC with TSV serialization. In such a condition where link bandwidth is asymmetric, utilizing limited bandwidth in a better way is more beneficial than taking short routing paths. To avoid performance collapsing on high traffic load, TSV ejection/injection scheme is used, and corresponding deadlock and livelock problems are solved in a simple, yet efficient way. Experiments show that our proposed network shows great performance while consuming minimal power. Compared to simple buffered routers, the proposed router shows 2.3% better throughput and 30.8% lower energy for synthetic traffic load. It also performs 43.3% faster, consumes 63.0% lower energy, and shows 4.8 times better energy efficiency for application based traffics. On real application benchmarks, it performs 30.0% faster than buffered routers with four virtual channels, consumes 78.6% lower energy consumption, and shows 6.67 times better energy efficiency.

Also, area cost of the proposed router is almost half of a simple buffered router and about four times smaller than a buffered router with virtual channels.

In the last part, we proposed a deadlock-free routing algorithm for 3-D NoC with partial vertical connections without using any VCs. Proofs are given and the experimental results show the benefits from the proposed routing algorithm in the view of energy efficiency or performance. Compared to the conventional routing algorithm (i.e., baseline elevator-first algorithm), the proposed algorithm shows improvement of 38.9% in energy consumption, or 6.9% in performance. Partially connected architectures are likely to be used because of the high cost of TSVs. However, as can be seen from the experimental results, its reduced vertical bandwidth can easily become a bottleneck. We believe that it can be mitigated by mapping tasks/data into nodes efficiently, or by using adaptive routing based on the proposed idea, which is our future work.

After decades of researches, the network-on-chip area is becoming mature and starting to be used to commercial products. However, there are still many problems unsolved, which we believe will become more significant in the future, due to the increased number of integrated cores in the system. Thus, we still need a lot of topics to be researched on network-on-chip. For example, there is not any 3D NoC in commercial use as far as our knowledge. Many problems such as thermal and routing have to be solved in order to realize them. We hope this thesis can give helpful insights to the readers.

# Bibliography

- [1] “International technology roadmap for semiconductors.”  
<http://www.itrs.net>.
- [2] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson, “A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS,” in *International Solid-State Circuits Conference Digest of Technical Papers*, pp. 108–109, 2010.
- [3] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, “Floorplan-aware automated synthesis of bus-based communication architectures,” in *Proceedings of the Design Automation Conference*, pp. 565–570.
- [4] J. Yoo, D. Lee, S. Yoo, and K. Choi, “Communication architecture synthesis of cascaded bus matrix,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 171–177, 2007.

- [5] L. Benini and G. De Micheli, “Networks on chips: A new SoC paradigm,” *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.
- [6] K. Goossens, J. Dielissen, and A. Radulescu, “Æthereal network on chip: Concepts, architectures, and implementations,” *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.
- [7] “Adapteva epiphany,” 2012. <http://www.adapteva.com/products/epiphany-ip/epiphany-architecture-ip/>.
- [8] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, “On-chip interconnection architecture of the tile processor,” *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.
- [9] D. Bertozzi and L. Benini, “Xpipes: A network-on-chip architecture for gigascale systems-on-chip,” *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, 2004.
- [10] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, “A 5-GHz mesh interconnect for a teraflops processor,” *IEEE Micro*, vol. 27, no. 5, pp. 51–61.
- [11] J. Hu and R. Marculescu, “Energy- and performance-aware mapping for regular NoC architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, 2005.
- [12] J. Kim, “Low-cost router microarchitecture for on-chip networks,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 255–266, 2009.

- [13] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, “Darknoc: Designing energy-efficient network-on-chip with multi-vt cells for dark silicon,” in *Proceedings of the Design Automation Conference*, pp. 1–6, 2014.
- [14] X. Wang, G. Gan, J. Manzano, D. Fan, and S. Guo, “A quantitative study of the on-chip network and memory hierarchy design for many-core processor,” in *Proceedings of the International Conference on Parallel and Distributed Systems*, pp. 689–696, 2008.
- [15] J. Yoo, S. Yoo, and K. Choi, “Multiprocessor system-on-chip designs with active memory processors for higher memory efficiency,” in *Proceedings of the Design Automation Conference*, pp. 806–811, 2009.
- [16] J. Heinlein, K. Gharachorloo, S. Dresser, and A. Gupta, “Integration of message passing and shared memory in the stanford FLASH multiprocessor,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 38–50, 1994.
- [17] A. Agarwal, R. Bianchini, D. Chaiken, F. Chong, K. Johnson, D. Kranz, J. Kubiawicz, B.-H. Lim, K. MacKenzie, and D. Yeung, “The MIT alewife machine,” *Proceedings of the IEEE*, vol. 87, no. 3, pp. 430–444, 1999.
- [18] C.-C. Kuo, J. Carter, R. Kuramkote, and M. Swanson, “ASCOMA: An adaptive hybrid shared memory architecture,” in *Proceedings of the International Conference on Parallel Processing*, pp. 207–216, 1998.
- [19] S. Tota, M. Casu, M. Ruo Roch, L. Rostagno, and M. Zamboni, “MEDEA: A hybrid shared-memory/message-passing multiprocessor NoC-based architecture,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 45–50, 2010.

- [20] Z. Yu, K. You, R. Xiao, H. Quan, P. Ou, Y. Ying, H. Yang, M. Jing, and X. Zeng, “An 800MHz 320mW 16-core processor with message-passing and shared-memory inter-core communication mechanisms,” in *International Solid-State Circuits Conference Digest of Technical Papers*, pp. 64–66, 2012.
- [21] J. Lee and K. Choi, “Memory-aware mapping and scheduling of tasks and communications on many-core SoC,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 419–424, 2012.
- [22] J. Lee, M.-K. Chung, Y.-G. Cho, S. Ryu, J. H. Ahn, and K. Choi, “Mapping and scheduling of tasks and communications on many-core soc under local memory constraint,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1748–1761, 2013.
- [23] J. Zhao, X. Dong, and Y. Xie, “An energy-efficient 3D CMP design with fine-grained voltage scaling,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 1–4, 2011.
- [24] L. Shang, L.-S. Peh, and N. K. Jha, “Dynamic voltage scaling with links for power optimization of interconnection networks,” in *Proceedings of the International Symposium on High-Performance Computer Architecture*, pp. 91–102, 2003.
- [25] A. K. Mishra, A. Yanamandra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, “RAFT: A router architecture with frequency tuning for on-chip networks,” *Journal of Parallel and Distributed Computing*, vol. 71, no. 5, pp. 625–640, 2011.
- [26] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff, “Energy characterization of a tiled architecture processor with on-chip networks,” in *Proceed-*

- ings of the International Symposium on Low Power Electronics and Design*, pp. 424–427, 2003.
- [27] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, “A 5-GHz mesh interconnect for a teraflops processor,” *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.
- [28] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha, “Thermal modeling, characterization and management of on-chip networks,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 67–78, 2004.
- [29] J. Lee, J. Ahn, K. Choi, and K. Kang, “THOR: Orchestrated thermal management of cores and networks in 3d many-core architectures,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 773–778, 2015.
- [30] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das, “A novel dimensionally-decomposed router for on-chip communication in 3D architectures,” in *Proceedings of the International Symposium on Computer Architecture*, pp. 138–149, 2007.
- [31] X. Wang, M. Palesi, M. Yang, Y. Jiang, M. Huang, and P. Liu, “Low latency and energy efficient multicasting schemes for 3D NoC-based SoCs,” in *International Conference on VLSI and System-on-Chip*, pp. 337–342, 2011.
- [32] A. Rahmani, K. Latif, K. R. Vaddina, P. Liljeberg, J. Plosila, and H. Tenhunen, “ARB-NET: A novel adaptive monitoring platform for stacked mesh 3D NoC architectures,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 413–418, 2012.

- [33] A.-M. Rahmani, K. Latif, K. R. Vaddina, P. Liljeberg, J. Plosila, and H. Tenhunen, "Congestion aware, fault tolerant, and thermally efficient inter-layer communication scheme for hybrid NoC-bus 3D architectures," in *Proceedings of the International Symposium on Networks on Chips*, pp. 65–72, 2011.
- [34] D. Park, S. Eachempati, R. Das, A. K. Mishra, Y. Xie, N. Vijaykrishnan, and C. R. Das, "MIRA: A multi-layered on-chip interconnect router architecture," in *Proceedings of the International Symposium on Computer Architecture*, pp. 251–261, 2008.
- [35] H. Sangki, "3D super-via for memory applications," in *Micro-Systems Packaging Initiative Packaging Workshop*, 2007.
- [36] G. Van der Plas, P. Limaye, I. Loi, A. Mercha, H. Oprins, C. Torregiani, S. Thijs, D. Linten, M. Stucchi, G. Katti, D. Velenis, V. Cherman, B. Vandevelde, V. Simons, I. De Wolf, R. Labie, D. Perry, S. Bronckers, N. Minas, M. Cupac, W. Ruythooren, J. Van Olmen, A. Phommahaxay, M. de Potter de ten Broeck, A. Opdebeeck, M. Rakowski, B. De Wachter, M. Dehan, M. Nelis, R. Agarwal, A. Pullini, F. Angiolini, L. Benini, W. Dehaene, Y. Travalay, E. Beyne, and P. Marchal, "Design issues and considerations for low-cost 3-D TSV IC technology," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 293–307, 2011.
- [37] W. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. Sule, M. Steer, and P. Franzon, "Demystifying 3D ICs: The pros and cons of going vertical," *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 498–510, 2005.
- [38] N. Miura, D. Mizoguchi, M. Inoue, K. Niitsu, Y. Nakagawa, M. Tago, M. Fukaishi, T. Sakurai, and T. Kuroda, "A 1 Tb/s 3 W inductive-coupling

- transceiver for 3D-stacked inter-chip clock and data link,” *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 111–122, 2007.
- [39] K. Niitsu, Y. Shimazaki, Y. Sugimori, Y. Kohama, K. Kasuga, I. Nonomura, M. Saen, S. Komatsu, K. Osada, N. Irie, T. Hattori, A. Hasegawa, and T. Kuroda, “An inductive-coupling link for 3D integration of a 90nm CMOS processor and a 65nm CMOS SRAM,” in *International Solid-State Circuits Conference Digest of Technical Papers*, pp. 480–481,481a, 2009.
- [40] S. Saito, Y. Kohama, Y. Sugimori, Y. Hasegawa, H. Matsutani, T. Sano, K. Kasuga, Y. Yoshida, K. Niitsu, N. Miura, T. Kuroda, and H. Amano, “MuCCRA-cube: A 3D dynamically reconfigurable processor with inductive-coupling link,” in *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp. 6–11, 2009.
- [41] J. Lee, M. Zhu, K. Choi, J. H. Ahn, and R. Sharma, “3D network-on-chip with wireless links through inductive coupling,” in *Proceedings of the International SoC Design Conference*, pp. 353–356, 2011.
- [42] S. Pasricha, “Exploring serial vertical interconnects for 3D ICs,” in *Proceedings of the Design Automation Conference*, pp. 581–586, 2009.
- [43] F. Dubois, A. Sheibanyrad, F. Petrot, and M. Bahmani, “Elevator-first: A deadlock-free distributed routing algorithm for vertically partially connected 3D-NoCs,” *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 609–615, 2013.
- [44] C. Fallin, C. Craik, and O. Mutlu, “CHIPPER: A low-complexity bufferless deflection router,” in *Proceedings of the International Symposium on High Performance Computer Architecture*, pp. 144–155, 2011.

- [45] Tiler, “The TILE-Gx<sup>TM</sup> processor family,” 2009.
- [46] J. Lee, D. Lee, S. Kim, and K. Choi, “Deflection routing in 3D network-on-chip with TSV serialization,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 29–34, 2013.
- [47] J. Lee, D. Lee, S. Kim, and K. Choi, “Deflection routing in 3D network-on-chip with limited vertical bandwidth,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 4, p. 50, 2013.
- [48] J. Lee and K. Choi, “A deadlock-free routing algorithm requiring no virtual channel on 3D-NoCs with partial vertical connections,” in *Proceedings of the International Symposium on Networks on Chip*, pp. 1–2, 2013.
- [49] J. Lee and K. Choi, “REDELf: An energy-efficient deadlock-free routing for 3-D NoCs with partial vertical connections,” *ACM Journal of Emerging Technologies*, vol. 12, no. 3, pp. 26:1–26:22, 2015.
- [50] U. Y. Ogras, J. Hu, and R. Marculescu, “Key research problems in noc design: A holistic perspective,” in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pp. 69–74, 2005.
- [51] S. Murali and G. De Micheli, “Bandwidth-constrained mapping of cores onto NoC architectures,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 896–901, IEEE Computer Society, 2004.
- [52] K. Srinivasan and K. S. Chatha, “A technique for low energy mapping and routing in network-on-chip architectures,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 387–392, ACM, 2005.

- [53] C.-L. Chou and R. Marculescu, “Contention-aware application mapping for network-on-chip communication architectures,” in *Proceedings of the International Conference on Computer Design*, pp. 164–169, 2008.
- [54] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel, “Exploring NoC mapping strategies: An energy and timing aware technique,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 502–507, 2005.
- [55] J. Hu and R. Marculescu, “Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 234–239.
- [56] R. Xu, R. Melhem, and D. Mosse, “Energy-aware scheduling for streaming applications on chip multiprocessors,” in *Proceedings of the Real-Time Systems Symposium*, pp. 25–38, 2007.
- [57] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal, “Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs,” in *Proceedings of the Design Automation Conference*, pp. 777–782, 2007.
- [58] K. Vivekanandarajah and S. Pilakkat, “Task mapping in heterogeneous MPSoCs for system level design,” in *Proceedings of the International Conference on Engineering of Complex Computer Systems*, pp. 56–65, 2008.
- [59] H. Yang and S. Ha, “ILP based data parallel multi-task mapping/scheduling technique for MPSoC,” in *Proceedings of the International SoC Design Conference*, pp. 134–137, 2008.

- [60] V. Suhendra, C. Raghavan, and T. Mitra, “Integrated scratchpad memory optimization and task scheduling for MPSoC architectures,” in *Proceedings of the International Conference on Compilers, Architecture and Synthesis of Embedded Systems*, pp. 401–410, 2006.
- [61] F. Angiolini, L. Benini, and A. Caprara, “Polynomial-time algorithm for on-chip scratchpad memory partitioning,” in *Proceedings of the International Conference on Compilers, Architecture and Synthesis of Embedded Systems*, pp. 318–326, 2003.
- [62] J. Lee and K. Choi, “Memory-aware mapping and scheduling of tasks and communications on many-core SoC,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 419–424, 2012.
- [63] J. Codina, J. Sanchez, and A. Gonzalez, “Virtual cluster scheduling through the scheduling graph,” in *Proceedings of the International Symposium on Code Generation and Optimization*, pp. 89–101, 2007.
- [64] A. Aleta, J. Codina, J. Sanchez, A. Gonzalez, and D. Kaeli, “AGAMOS: A graph-based approach to modulo scheduling for clustered microarchitectures,” *IEEE Transactions on Computers*, vol. 58, no. 6, pp. 770–783, 2009.
- [65] P. Shivakumar and N. P. Jouppi, “CACTI 3.0: An integrated cache timing, power, and area model,” tech. rep., 2001/2, Compaq Computer Corporation, 2001.
- [66] “ARM9 processor family.”  
<http://mobile.arm.com/products/processors/classic/arm9/>.
- [67] “SoC Designer Plus.” <http://carbondesigntsystems.com/>.
- [68] M. Snir, *MPI - The Complete Reference*. MIT Press, 1998.

- [69] S. Tota, M. Casu, P. Motto, M. Ruo Roch, and M. Zamboni, “The NoCRay graphic accelerator: A case-study for MP-SoC network-on-chip design methodology,” in *Proceedings of the International Symposium on System-on-Chip*, pp. 1–4, 2007.
- [70] T. Ye, L. Benini, and G. De Micheli, “Packetized on-chip interconnect communication analysis for MPSoC,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 344–349, 2003.
- [71] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, “ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 423–428, 2009.
- [72] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff, “Energy characterization of a tiled architecture processor with on-chip networks,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 424–427, 2003.
- [73] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, “McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 469–480, 2009.
- [74] M.-T. Yang, R. Kasturi, and A. Sivasubramaniam, “A pipeline-based approach for scheduling video processing algorithms on NOW,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 2, pp. 119–130, 2003.
- [75] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, “Trade-offs in the design of a router

- with both guaranteed and best-effort services for networks on chip,” *IEE Proceedings—Computers and Digital Techniques*, vol. 150, no. 5, pp. 294–302, 2003.
- [76] Y. Salah, M. Zeghid, I. E. Bennour, and R. Tourki, “A scheduling approach for packet-switched on-chip networks,” in *Proceedings of the International Conference on Communications, Computing and Control Applications*, pp. 1–8, 2011.
- [77] M. R. Garey and D. S. Johnson, *Computers and Intractability*, vol. 29. WH Freeman, 2002.
- [78] K.-H. Han and J.-H. Kim, “Quantum-inspired evolutionary algorithm for a class of combinatorial optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 580–593, 2002.
- [79] Y. Ahn, K. Han, G. Lee, H. Song, J. Yoo, K. Choi, and X. Feng, “SoC-DAL: System-on-chip design AcceLerator,” *ACM Transactions on Design Automation of Electronic System*, vol. 13, no. 1, pp. 17:1–17:38.
- [80] H. Yang and S. Ha, “Pipelined data parallel task mapping/scheduling technique for MPSoC,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 69–74.
- [81] R. P. Dick, D. L. Rhodes, and W. Wolf, “TGFF: Task graphs for free,” in *Proceedings of the International Workshop on Hardware/Software Codesign*, pp. 97–101, 1998.
- [82] P. Moscato *et al.*, “On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms,” *Caltech concurrent computation program, C3P Report*, vol. 826, 1989.

- [83] B. R. Rau, “Iterative modulo scheduling: An algorithm for software pipelining loops,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 63–74, ACM, 1994.
- [84] K. Chatha and R. Vemuri, “Hardware-software partitioning and pipelined scheduling of transformative applications,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 193–208, 2002.
- [85] C. Sechen and A. Sangiovanni-Vincentelli, “The TimberWolf placement and routing package,” *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 510–522, 1985.
- [86] K. Skadron, T. Abdelzaher, and M. R. Stan, “Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management,” in *Proceedings of the International Symposium on High-Performance Computer Architecture*, pp. 17–28, 2002.
- [87] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, “Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 335–346, 2010.
- [88] A. Snaveley and D. M. Tullsen, “Symbiotic jobscheduling for a simultaneous multithreading processor,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 234–244, 2000.
- [89] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *Proceedings of the International Symposium on High-Performance Computer Architecture*, pp. 171–182, 2001.

- [90] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploration,” in *Proceedings of the International Symposium on Computer Architecture*, pp. 78–88, 2006.
- [91] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph, “Three-dimensional chip-multiprocessor run-time thermal management,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1479–1492, 2008.
- [92] X. Zhou, J. Yang, Y. Xu, Y. Zhang, and J. Zhao, “Thermal-aware task scheduling for 3D multicore processors,” *Journal of Parallel and Distributed Computing*, vol. 21, no. 1, pp. 60–71, 2010.
- [93] K. Kang, J. Kim, S. Yoo, and C.-M. Kyung, “Runtime power management of 3-D multi-core architectures under peak power and temperature constraints,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 6, pp. 905–918, 2011.
- [94] C.-H. Chao, K.-Y. Jheng, H.-Y. Wang, J.-C. Wu, and A.-Y. Wu, “Traffic- and thermal-aware run-time thermal management scheme for 3D NoC systems,” in *Proceedings of the International Symposium on Networks on Chips*, pp. 223–230, 2010.
- [95] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras, “In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches,” in *Proceedings of the International Symposium on Networks on Chips*, pp. 43–50, 2012.
- [96] F. Hameed, M. Faruque, and J. Henkel, “Dynamic thermal management in 3D multi-core architecture through run-time adaptation,” in *Proceedings of the Design, Automation and Test in Europe*, pp. 1–6, 2011.

- [97] V. Hanumaiah and S. Vrudhula, “Energy-efficient operation of multi-core processors by DVFS, task migration and active cooling,” *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 349–360, 2014.
- [98] J. A. Butts and G. S. Sohi, “A static power model for architects,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 191–201, 2000.
- [99] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., 1997.
- [100] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 52, 2011.
- [101] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, “HotSpot: A compact thermal modeling methodology for early-stage VLSI design,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.
- [102] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, “DSENT—a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling,” in *Proceedings of the International Symposium on Networks on Chip*, pp. 201–210, 2012.
- [103] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, “Predicting performance impact of DVFS for realistic memory systems,” in *Proceedings of the International Symposium on Microarchitecture*, pp. 155–165, 2012.

- [104] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.
- [105] T. Moscibroda and O. Mutlu, “A case for bufferless routing in on-chip networks,” in *Proceedings of the International Symposium on Computer Architecture*, pp. 196–207, 2009.
- [106] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu, “MinBD: Minimally-buffered deflection routing for energy-efficient interconnect,” in *Proceedings of the International Symposium on Networks on Chips*, pp. 1–10.
- [107] R. Ausavarungnirun, C. Fallin, X. Yu, K. K.-W. Chang, G. Nazario, R. Das, G. H. Loh, and O. Mutlu, “Design and evaluation of hierarchical rings with deflection routing,” in *Proceedings of the International Symposium on Computer Architecture and High Performance Computing*, pp. 230–237, 2014.
- [108] M. Zhu, J. Lee, and K. Choi, “An adaptive routing algorithm for 3d mesh NoC with limited vertical bandwidth,” in *Proceedings of the International Conference on VLSI and System-on-Chip*, pp. 18–23, 2012.
- [109] D. H. Kim and S. K. Lim, “Through-silicon-via-aware delay and power prediction model for buffered interconnects in 3D ICs,” in *Proceedings of the International Workshop on System Level Interconnect Prediction*, pp. 25–32, 2010.
- [110] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, “Evaluating bufferless flow control for on-chip networks,” in *Proceedings of the International Symposium on Networks on Chips*, pp. 9–16, 2010.

- [111] W. J. Dally, “Virtual-channel flow control,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.
- [112] J. Ouyang, J. Xie, M. Poremba, and Y. Xie, “Evaluation of using inductive/capacitive-coupling vertical interconnects in 3D network-on-chip,” in *Proceedings of the International Conference on Computer-Aided Design*, pp. 477–482, 2010.
- [113] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, “A NoC traffic suite based on real applications,” in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 66–71, 2011.
- [114] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in *Proceedings of the International Symposium on Computer Architecture*, pp. 24–36, 1995.
- [115] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The PARSEC benchmark suite: Characterization and architectural implications,” in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 72–81, 2008.
- [116] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 52, 2011.
- [117] C. Kim, D. Burger, and S. W. Keckler, “An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 211–222, 2002.

- [118] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, and V. Narayanan, “Design and management of 3D chip multiprocessors using network-in-memory,” in *Proceedings of the International Symposium on Computer Architecture*, pp. 130–141, 2006.
- [119] N. Dahir, R. Al-Dujaily, A. Yakovlev, P. Missailidis, and T. Mak, “Deadlock-free and plane-balanced adaptive routing for 3D networks-on-chip,” in *Proceedings of the International Workshop on Network on Chip Architectures*, pp. 31–36, 2012.
- [120] F. Darve, A. Sheibanyrad, P. Vivet, and F. Petrot, “Physical implementation of an asynchronous 3D-NoC router using serial vertical links,” in *Proceeding of the IEEE Computer Society Annual Symposium on VLSI*, pp. 25–30, 2011.
- [121] S. Pasricha, “A framework for TSV serialization-aware synthesis of application specific 3D networks-on-chip,” in *Proceedings of the International Conference on VLSI Design*, pp. 268–273, 2012.
- [122] B. Akesson, P.-C. Huang, F. Clermidy, D. Dutoit, K. Goossens, Y.-H. Chang, T.-W. Kuo, P. Vivet, and D. Wingard, “Memory controllers for high-performance and real-time MPSoCs: Requirements, architectures, and future trends,” in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pp. 3–12, 2011.
- [123] C. Liu, L. Zhang, Y. Han, and X. Li, “Vertical interconnects squeezing in symmetric 3D mesh network-on-chip,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 357–362, 2011.

- [124] T. C. Xu, P. Liljeberg, and H. Tenhunen, “A study of through silicon via impact to 3D network-on-chip design,” in *Proceedings of the International Conference On Electronics and Information Engineering*, 2010.
- [125] Y. Xu, Y. Du, B. Zhao, X. Zhou, Y. Zhang, and J. Yang, “A low-radix and low-diameter 3D interconnection network design,” in *Proceedings of the International Symposium on High Performance Computer Architecture*, pp. 30–42, 2009.
- [126] N. Kapadia and S. Pasricha, “A power delivery network aware framework for synthesis of 3D networks-on-chip with multiple voltage islands,” in *Proceedings of the International Conference on VLSI Design*, pp. 262–267, 2012.
- [127] C. Seiculescu, S. Murali, L. Benini, and G. De 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*, pp. 9–14, 2009.
- [128] S. Yan and B. Lin, “Design of application-specific 3D networks-on-chip architectures,” in *3D Integration for NoC-based SoC Architectures*, pp. 167–191, Springer, 2011.
- [129] W. J. Dally and C. L. Seitz, “Deadlock-free message routing in multiprocessor interconnection networks,” *IEEE Transactions on Computers*, vol. 100, no. 5, pp. 547–553, 1987.
- [130] O. Lysne, T. Skeie, S.-A. Reinemo, and I. Theiss, “Layered routing in irregular networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 1, pp. 51–65, 2006.

## 초록

지난 수십 년간 이어진 반도체 기술의 향상은 매니 코어의 시대를 가져다 주었다. 우리가 일상 생활에 쓰는 데스크톱 컴퓨터조차도 이미 수 개의 코어를 가지고 있으며, 수백 개의 코어를 가진 칩도 상용화되어 있다. 이러한 많은 코어들 간의 통신 기반으로서, 네트워크-온-칩(NoC)이 새로이 대두되었으며, 이는 현재 많은 연구 및 상용 제품에서 널리 사용되고 있다. 그러나 네트워크-온-칩을 매니 코어 시스템에 사용하는 데에는 여러 가지 문제가 따르며, 본 논문에서는 그 중 몇 가지를 풀어내고자 하였다.

본 논문의 두 번째 챕터에서는 NoC 기반 매니코어 구조에 작업을 할당하고 스케줄하는 방법을 다루었다. 매니코어에의 작업 할당을 다룬 논문은 이미 많이 출판되었지만, 본 연구는 메시지 패싱과 공유 메모리, 두 가지의 통신 방식을 고려함으로써 성능과 에너지 효율을 개선하였다. 또한, 본 연구는 역방향 의존성을 가진 작업 그래프를 스케줄하는 방법 또한 제시하였다.

3차원 적층 기술은 높아진 전력 밀도 때문에 열 문제가 심각해지는 등, 여러 가지 도전 과제를 내포하고 있다. 세 번째 챕터에서는 DVFS 기술을 이용하여 열 문제를 완화하고자 하는 기술을 소개한다. 각 코어와 라우터가 전압, 작동 속도를 조절할 수 있는 구조에서, 가장 높은 성능을 이끌어 내면서도 최대 온도를 넘어서지 않도록 한다.

세 번째와 네 번째 챕터는 조금 다른 측면을 다룬다. 3D 적층 기술을 사용할 때, 층간 통신은 주로 TSV를 이용하여 이루어진다. 그러나 TSV는 일반 wire보다 훨씬 큰 면적을 차지하기 때문에, 전체 네트워크에서의 TSV 개수는 제한되어야 할 경우가 많다. 이 경우에는 두 가지 선택지가 있는데, 첫째는 각 층간 통신 채널의 대역폭을 줄이는 것이고, 둘째는 각 채널의 대역폭을 유지하되 일부 노드만 층간 통신이 가능한 채널을 제공하는 것이다. 우리는 각각의 경우에 대하여 라우팅

알고리즘을 하나씩 제시한다.

첫 번째 경우에 있어서는 deflection 라우팅 기법을 사용하여 층간 통신의 긴 지연 시간을 극복하고자 하였다. 층간 통신을 균등하게 분배함으로써, 제시된 알고리즘은 개선된 지연 시간을 보이며 라우터 버퍼의 제거를 통한 면적 및 에너지 효율성 또한 얻을 수 있다. 두 번째 경우에는 층간 통신 채널을 선택하기 위한 몇 가지 규칙을 제시한다. 약간의 라우팅 자유도를 희생함으로써, 제시된 알고리즘은 기존 알고리즘의 가상 채널 요구 조건을 제거하고, 결과적으로는 성능 또는 에너지 효율의 증가를 가져 온다.

**주요어:** 네트워크-온-칩, 스케줄링, 작업 할당, 온도 관리, 라우팅, 3D 적층

**학번:** 2011-30251