UNIVERSITY OF CALIFORNIA, SAN DIEGO

Synthesis of Application-Specific On-Chip Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical and Computer Engineering

by

Shan Yan

2009

DEDICATION

To my husband Bo Pan, my parents Gongrong Yan and Yuying Shi, and my son George.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I feel fortunate to have received a lot of help from so many people during my PhD study in UCSD.

First, I own my deep gratitude to my great advisor, Professor Bill Lin, for his patience, insight, enlightenment, and help. He has set himself a very professional example for me to perform research, as well as deal with challenges we are facing everyday. His respect for innovation and hard work are part of a life-time gift that I can benefit in my future career. Besides those, I am also impressed by his confidence, diligence, passion, persistence and smartness. Without him, I could have not gone through all the difficulties and achieved such a proud achievement in my life. It is indeed a great privilege for me to study under his guidance.

Second, I am grateful to my proposal and dissertation committee members, Professor Sujit Dey, Professor Curt Schurgers, Professor Amin Vahdat, and Professor Michael B. Taylor for their time and their reviews and suggestions of this dissertation.

I would like to thank all colleagues from our research group, as well as all of my friends, including but not limited to Yanhua Mao, Chengmo Yang, Dan Liu, Zheng Wu, Haichang Sui and Wenyi Zhang, thank all of them for their generous help in research and companionship in these years. We have enjoyed the colorful life in La Jolla these years. Jerry Chou, Chia-wei Chang, Hao Wang, Siddhartha Saha and Rohit S. Ramanujam, thank them for making the group a friendly and fun place to work in.

I owe too much to my parents, Gongrong Yan and Yuying Shi, for their endless love and unlimited support for all my educations and for helping me understand and explore the rich and colorful outer world. I would be nothing without them in every conceivable way. I wish they can always feel proud of me.

I feel blessed to have my adorable son, George Yan Pan, for witnessing the height I have reached.

Last and most importantly, I want to thank my great and handsome husband Bo Pan. I know these six years are too much for you. There is no such single word that can express my deepest gratitude for your love, faith, understanding, tolerance, and support.

Chapter 2 is in part a reprint of the materials in the papers: Shan Yan, Bill Lin, "Custom Networks-on-Chip Architectures with Multicast Routing", *IEEE Transactions on*

VLSI Systems, volume: 17, issue: 3, on pages: 342-355, March 2009, and Shan Yan, Bill Lin, “Joint Multicast Routing and Network Design Optimization for Networks-on-Chip”, *IET Computers and Digital Techniques*, accepted for publication, 2009. Chapter 3 is in part a reprint of the material in the papers: Shan Yan, Bill Lin, “Custom Networks-on-Chip Architectures with Multicast Routing”, *IEEE Transactions on VLSI Systems*, volume: 17, issue: 3, on pages: 342-355, March 2009, and Shan Yan, Bill Lin, “Application-specific Network-on-Chip architecture synthesis based on set partitions and Steiner Trees”, *13th Asia and south Pacific Design automation conference (ASP-DAC 2008)*, 2008: 277-282. Chapter 4 is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Joint Multicast Routing and Network Design Optimization for Networks-on-Chip”, *IET Computers and Digital Techniques*, accepted for publication, 2009. Chapter 5 is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Design of Application-Specific 3D Network-on-Chip Architectures”, *The International Conference on Computer Design (ICCD 2008)*, 2008 Chapter 6 is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Design of Application-Specific On-Chip Networks for Multiple Usage Scenarios", submitted to *IEEE Embedded Systems Letters*, 2009 Chapter 7 is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Custom Networks-on-Chip Architectures with Multicast Routing”, *IEEE Transactions on VLSI Systems*, volume: 17, issue: 3, on pages: 342-355, March 2009. The dissertation author was the primary author of all these papers.

VITA

2000	B. S. in Electronic Engineering, Tsinghua University, Beijing, China
2003	M. S. in Electronic Engineering, Tsinghua University, Beijing, China
2009	Ph. D. in Electrical and Computer Engineering, University of California, San Diego

PUBLICATIONS

Shan Yan, Bill Lin, "Joint Multicast Routing and Network Design Optimization for Networks-on-Chip", *IET Computers and Digital Techniques*, accepted for publication, 2009.

Shan Yan, Bill Lin, "Custom Networks-on-Chip Architectures with Multicast Routing", *IEEE Transactions on VLSI Systems*, volume: 17, issue: 3, on pages: 342-355, March 2009.

Shan Yan, Bill Lin, "Design of Application-Specific 3D Network-on-Chip Architectures", *The International Conference on Computer Design (ICCD 2008)*, 2008.

Shan Yan, Bill Lin, "Application-specific Network-on-Chip architecture synthesis based on set partitions and Steiner Trees", *13th Asia and South Pacific Design Automation Conference (ASP-DAC 2008)*, 2008: 277-282

Shan Yan, Bill Lin, "Stream Execution on Embedded Wide-Issue Clustered VLIW Architectures", *EURASIP Journal on Embedded Systems*, vol. 2008, Article ID 516240, 9 pages, 2008

Shan Yan, Bill Lin, "Stream execution on wide-issue clustered VLIW architectures",*2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'07)*, 2007: 158-160

Shan Yan, Bill Lin, "Optimized Custom Network Design for Multiple Traffic Profiles", submitted to *NOCS 2009*

Shan Yan, Bill Lin, "Design of Application-Specific On-Chip Networks for Multiple Usage Scenarios", submitted to *IEEE Embedded Systems Letters, 2009*

ABSTRACT OF THE DISSERTATION

Synthesis of Application-Specific On-Chip Networks

by

Shan Yan
Doctor of Philosophy in Electrical and Computer Engineering

University of California San Diego, 2009

Professor Bill Lin, Chair

Networks-on-Chip (NoC) has been proposed as a scalable solution to the global communication challenges in nanoscale System-on-Chip designs. The use of NoCs with standardized interfaces facilitates the reuse of previously-designed and third-party-provided modules in new designs. Besides design and verification benefits, NoCs have also been advocated to address increasingly daunting clocking, signal integrity, and wire delay challenges.

In this thesis, we present design methods for synthesizing NoC architectures that are optimized for specific applications. We first present a novel design flow that integrates floorplanning, NoC architecture synthesis, RTL generation, and detailed RTL design. The proposed design flow is very flexible in that it allows for different user-defined objectives and constraints. The proposed design flow also supports both unicast and multicast traffic.

We then present two approaches to the NoC synthesis problem. The first approach is based on flow-set partitioning and Steiner-tree construction. In this approach, the problem is decomposed into the inter-related steps of finding a good flow-set partition, deriving a good physical network topology for each group using Steiner-tree-based algorithms, and providing an optimized implementation for the derived topologies. The second approach is based on a rip-up and re-route formulation that successively improves upon an intermediate solution by “ripping” out a flow and freeing up any network resources occupied by it, then “re-routing” the ripped-up flow over the remaining network. To consider multicast flows, the re-routing step is formulated as a minimum directed spanning tree problem.

While both approaches are effective, we found that the rip-up and re-route approach is generally better. Therefore, we chose to extend this approach to consider two additional design dimensions. First, we describe extensions to consider 3D-NoC synthesis. This is motivated by the increasing viability of 3D integration that has opened new opportunities for chip design innovations. To support 3D-NoC synthesis, we propose accurate power and delay models for 3D-wires with through-silicon-vias. Second, we describe extensions to support multiple traffic profiles, which are useful for design applications that support multiple usage scenarios, each with its own traffic profile.

Finally, all proposed algorithms have been integrated into a software package called ARIES.

Chapter 1

Introduction

1.1 Limitations of Traditional On-Chip Communication Architectures

Information explosion has generated abundant data for computer systems to process. To accommodate this ever-increasing data growth, there is an ever-increasing need for more computing power in these systems to perform real-time data processing. This has naturally led to much more complicated Systems-on-Chip (SoC) designs. Thanks to advances in modern semiconductor technology, billions of transistors can now be integrated onto the same silicon device, enabling the integration of hundreds or thousands of function modules with large amounts of embedded memory to perform very sophisticated tasks. These function modules can be CPU or DSP cores, video streaming processors, security processors, high-bandwidth interfaces, etc [1]. However, the integration of many function modules has also created large volumes of on-chip traffic in many designs, making highly-efficient on-chip communication architectures necessary. Unfortunately, as mainstream semiconductor processes scale down from 180-nm to 45- or 32-nm technology, the architectures and design methodologies for on-chip communication have not kept pace. Increasing interconnect delays and power consumption of on-chip communication designs have become major performance bottlenecks in many SoC designs, largely hindering the performance of these systems.

Currently, bus-based architectures are still widely used as the dominant underly-

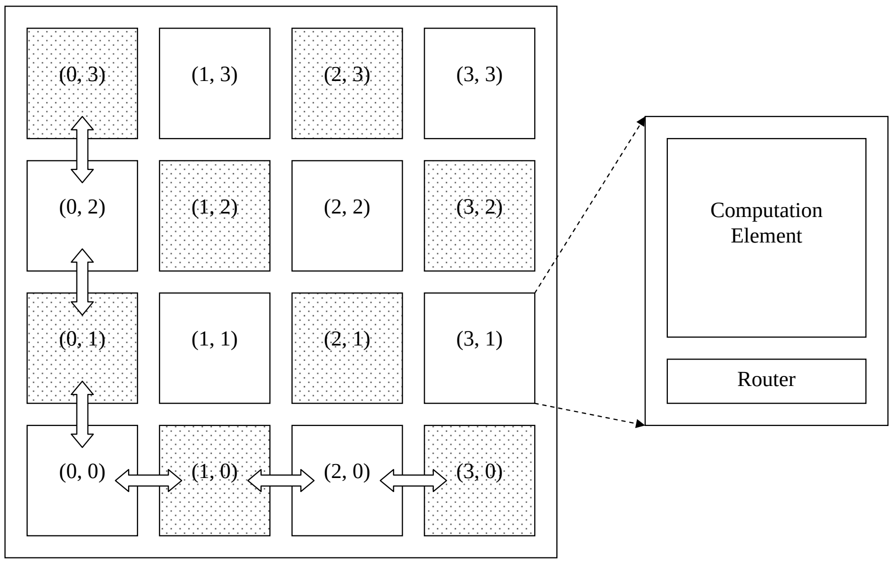


Figure 1.1: Regular tile-base NoC architecture [9]

ing fabric for supporting on-chip communications. There are many different bus-based architecture designs that have been used in various modern SoCs to meet different communication requirements, including several state-of-the-art bus architectures such as the AMBA multi-layer bus [4], STBus [5] and SonicsMX [6]. These latest architectures use multiple parallel buses to support multiple speeds for different levels of communication requirement. Communications between buses are usually through bridges. Despite many successful bus-based SoC designs that been developed in the past, many designers also recognize that bus-based architectures are inherently not scalable.

1.2 The NoC Approach

To address the global communication challenges in nanoscale SoC designs, Network-on-Chip (NoC) architectures have been proposed as a scalable solution [9, 10, 11]. NoCs are analogous to modern telecommunication networks that use packet-switching to relay traffic from any source node to any destination node with the help of routers within the network. In a NoC system, modules such as processor cores, embedded memories, and specialized Intellectual Property (IP) blocks exchange data by using a network as a “public transportation” sub-system.

An example is shown in Figure 1.1 to illustrate the fundamental principle of how an NoC works. The chip consists of a 2D mesh of tiles. Neighboring tiles are interconnected through direct links. Each tile may correspond to a general-purpose processor, a DSP core, a video streaming processor, an embedded memory, or an external I/O interface, etc. Please note Figure 1.1 is just a function level diagram. In general, the tiles may have very different physical sizes and shapes. To facilitate inter-tile communications, a router is embedded in each tile, and the interconnection of routers forms a global network. Thus, instead of running potentially long global wires or buses across the chip to facilitate communications between spatially distant tiles, inter-tile communication can be achieved simply by routing packets over the NoC. In comparison with bus-based architectures, NoC architectures offer the following benefits:

1. **Improved Efficiency**

   The NoC approach improves silicon efficiency by optimizing and distributing all on-chip communication resources in a systematic way. Virtual communication paths can be formed through the network over links and routers. This avoids the need for constructing permanent dedicated long wires that might be infrequently used. Precious silicon real estate can be greatly saved by sharing network resources.

2. **Improved Parallelism and Scalability**

   Unlike a global shared bus approach where all communications must be serialized to occur one at a time, a well-designed NoC architecture provides improved parallelism since data exchange can occur simultaneously over different parts of network. The NoC approach is also scalable in that the size of the network and the dimensioning of the network resources can grow with the design.

3. **Reduced Complexity**

   The use of NoCs with standardized interfaces facilitates the reuse of previously-designed and third-party-provided modules in new designs (e.g. processor cores). The NoC approach provides a clean separation between computation and communication, which also simplifies the verification problem as interactions between modules can be verified at a higher network transactions level rather than at a detailed wiring level. Modules can be largely verified separately.

4. **Improved Predictability**

   Unlike conventional VLSI approaches with ad-hoc global wirings, NoCs can be designed in a much more predictable way. Network interfaces and router microarchitectures can largely be designed and verified independently, and signal integrity and wiring delay issues can largely be isolated to the network links that interconnect the routers and interfaces.

1.3 Design Challenges of NoC Synthesis

NoC architectures can be designed as regular or custom network topologies. Regular topologies, such as mesh or folded-torus networks, have been successfully employed in a number of tile-based chip-multiprocessor projects, e.g. [15, 16], which are appropriate because of processor homogeneity and application traffic variability. On the other hand, for custom SoC applications, the design challenges are different in terms of varied module sizes, irregularly spread module locations, and different communication data rate requirements. Therefore, a custom network architecture optimized to the needs of the application is more appropriate. This synthesis problem is the focus of this thesis.

The NoC synthesis problem is challenging for a number of reasons. First, for a large complex SoC design, an optimal solution will likely involve multiple networks since each module will likely communicate only with a small subset of modules. Therefore, a single network that spans all nodes is often unnecessary. Part of the synthesis problem is to partition cores or the specified communication flows into groups, and connect each group of cores to the same router or derive a separate optimal physical topology for each group of flows so that they can share network resources. It is hard to decide which cores and flows should be partitioned into the same group. In general, cores may be grouped together and connect to the same routers even though they do not have flows that share common sources or destinations because such cores may be able to beneficially share common intermediate network resources. Also, it is hard to decide on the partition sizes beforehand, namely whether a design with a few large routers would be more cost effective than a design with many smaller routers.

Second, besides deciding on the partitioning of cores and flows, our synthesis problem must also decide on the physical network topology for each partition and on the connectivity between them. The physical network topology for each partition must be dimensioned properly to support the required traffic demands.

Finally, depending on the optimization goals and the implementation backend, the appropriate cost function may be quite complex. For example, when considering power minimization as the NoC design goal, both leakage power and dynamic switching power need to be taken into account. It is well-known that leakage power is becoming increasingly dominant [30, 32]. In the on-chip networks studied in [32], leakage power represented only about 0.6% and 1.8% of the total power consumption at 180nm and 100nm, respectively. However, leakage power increased to a hefty 32% at 70nm. High-performance microprocessor studies show even a much larger leakage power component [30]. Therefore, it is important to properly account for leakage power when adding routers and network links to the synthesized architecture. However, when considering leakage power, the cost function may need to account for possibly discrete cost increments of links and routers whereas dynamic switching power may be best modeled as a function of cumulative data rates. In general, the incorporation of accurate power and delay models in the NoC design flow is critical.

1.4 Related Work

1.4.1 Regular NoC architecture design

The NoC design problem has received considerable attention in the literature. Towles and Dally [9] and Benini and De Micheli [17] motivated the NoC paradigm. In [29], Bertozzi et al. addressed the complementary problem of providing custom network architecture instantiations.

NoC architectures can be designed as regular and custom network topologies. Regular topologies, such as mesh or folded-torus networks, have been successfully employed in a number of tile-based chip-multiprocessor projects, which are appropriate because of processor homogeneity and application traffic variability. For example, MIT's RAW processor [15] and UT Austin's TRIP architecture [16] are both based on the two dimensional mesh topologies.

For the custom SoC applications, several existing NoC solutions have addressed the mapping problem of different applications to a regular mesh-based NoC architecture [18, 19, 20, 21]. In [18] and [19], Lei and Kumar and Ascia et al. presented different genetic algorithms for the mapping of computation cores on to mesh-based NoC architectures. Hu and Marculescu [20] proposed a branch-and-bound algorithm for the mapping of computation cores on to mesh-based NoC architectures. Murali et al. [21] described a fast algorithm for mesh-based NoC architectures that considers different routing functions, delay constraints, and bandwidth requirements.

However, the regular NoC topologies may not be appropriate for the custom SoC applications because of their different properties such as the varied module sizes, irregularly spread module locations and different communication data rate requirements etc. Therefore, the custom network architecture optimized to the needs of the application is more appropriate.

1.4.2 Custom NoC architecture design

On the problem of designing custom NoC architectures without assuming an existing network architecture, a number of techniques have been proposed [22, 23, 24, 25, 27, 28]. Pinto et al. [24] presented techniques for the constraint-driven communication architecture synthesis of point-to-point links by using heuristic-based k-way merging. Their technique is limited to topologies with specific structures that have only two routers between each source and sink pair. Ogras et al. [22, 23] proposed graph decomposition and long link insertion techniques for application-specific NoC architectures.

Srinivasan et al. [25, 27] presented NoC synthesis algorithms that consider system-level floorplanning. They formulated NoC synthesis as a mixed integer linear programming problem. Their objective was to minimize the power consumption while satisfying performance constraints. Their NoC architecture is derived from the slicing structure of their floorplanning step by means of a channel intersection graph, where router locations are restricted to corners of cores and links run around cores. In addition, as stated in [25, 26, 27], their power minimization problem is one of minimizing the total traffic flowing through the routers of the derived NoC architecture, which corresponds well to process technologies where dynamic switching power dominates, but not necessarily when leakage power is substantial.

Murali et al. [28] presented an innovative deadlock-free NoC synthesis flow with detailed backend integration that also considers the floorplanning process. The proposed approach is based on the min-cut partitioning of cores to routers. However, their method does not consider topologies where there may be intermediate routers that do not directly connect to cores. Custom topologies that contain such intermediate routers may be useful for the sharing of network resources.

1.4.3 Multicast

Multicasting in wormhole-switched networks has been explored in the context of chip multiprocessors based on the methods in parallel machines for supporting cache coherency, acknowledgement collection, and synchronization, etc [68, 69]. In the NoC works of [79, 80], they have reported that multicast service can be implemented in their NoC architectures. However, the methods for providing multicast routing and services have not been presented in details. In [75], a novel multicast scheme in wormhole-switched NoCs using a connection-oriented technique to realize QoS-aware multicasting in a best-effort network was proposed to support SoC applications. In [81], a router architecture supporting unicast and multicast services was proposed using a mechanism for managing broadcast-flows so that the communication links in an on-chip network can be shared. In [82], the dual-path multicast algorithm, used in multicomputers, was adapted to wormhole-switched NoCs to support deadlock-free multicast routing.

1.4.4 3D NoC design

Research in 3D NoC is only emerging recently. Several works have been done in the area of 3D floorplanning and 3D placement and routing. Cong et al. [92, 96] proposed thermal-driven design flows for 3D ICs including 3D floorplanning and 3D placement and routing algorithms. In [93], thermal effect was formulated as another force in a force-directed approach to direct the placement procedure. In [94], floorplanning is done by accounting for the effects of the interconnect power consumption in estimating the peak temperatures. In [97], a thermal-aware Steiner routing algorithm for 3D ICs is proposed by constructing a delay-oriented Steiner tree under a given thermal profile in the first step and then conducting the refinement by repositioning the through-silicon vias for further thermal optimization.

On the problem of designing NoC architectures for 3D ICs, current literature has focused on regular 3D mesh NoC architectures [95, 98, 99, 100], which is appropriate for regular 3D processor designs [88, 89, 91]. Addo-Quaye [95] presented an algorithm for the thermal-aware mapping and placement of 3D NoCs including regular mesh topologies. Li et al. [89] proposed a similar 3D NoC topology targeting multi-processor systems by employing a bus structure for communications between different device layers. In [98], various possible topologies for 3D NoCs are presented and analytic models for zero-load latency and power consumption of these networks are described. However, all these 3D topologies are based on regular 3D mesh networks. To the best of our knowledge, a fully automated application-specific 3D-NoC synthesis solution has remained an open problem.

1.4.5 Multiple usage scenarios

Increasingly, multiprocessor SoCs are designed to support different usage scenarios since such SoC designs may be employed in different products or in products with different operation modes. For these embedded applications, the NoC must be designed to satisfy the communication characteristics and performance constraints of *all* usage scenarios considered. As discussed in greater details above, most prior work on custom NoC synthesis has only considered a *single* traffic profile in their formulations [24, 29, 22, 23, 25, 28, 57, 58]. Several papers have considered support for different usage scenarios [59, 60]. However, these papers assume the network topology is either *given* or is *regular*. Therefore, these works do not directly address our goal of synthesizing an optimized custom NoC architecture that supports multiple usage scenarios.

1.4.6 Deadlock considerations

Finally, deadlock-free routing is an important consideration for the correct operation of custom NoC architectures. The problem is very well studied and analyzed in the literature in the area of parallel computer systems and multiprocessor systems. In [66], Dally and Seitz proposed a necessary and sufficient condition for deterministic deadlock-free routing using the concept of a channel dependency graph. For general multiprocessor systems that can be programmed to run different applications, or in the case when adaptive routing is used, the problem is complicated by the challenge that the flows and routing paths are not necessarily known in advance [64, 65, 67]. In [67], Duato proposed a necessary and sufficient condition for adaptive deadlock-free routing. In [68], Lin et al. presented a deadlock-free dual-path routing algorithm for multicast wormhole routing for multicomputers adopting 2D-mesh and hypercube topologies.

A comprehensive survey on methods for handling message-dependent deadlocks in parallel computer systems is given in [70]. In contrast to the computer networks and multiprocessor environments studied in this work, NoC storage and computation resources are relatively more restricted, and the protocol stack is entirely implemented in hardware. Hence, design constraints and optimization goals are fundamentally different.

Many NoCs [71, 72, 73] break request-response dependencies by introducing separate physical networks for the two message types. Virtual networks, instead of physical, are used in [74, 75] to avoid deadlock in a higher-order configuration protocol and a forwarding multicast protocol, respectively. All these solutions are protocol-specific and none addresses the dependencies that can arise when IPs have both master and slave ports.

The possibility of considering message types in the topology synthesis was explored in [76]. This work presents a methodology that tailors the NoC to a particular application behavior while taking message-dependent deadlocks into account. Deadlocks are avoided when selecting the routes for flows using the turn prohibition algorithm presented in [77, 78].

1.5 Thesis Contributions

In this thesis, we present methodologies and algorithms for the design and synthesis of custom Networks-on-Chip (NoC) architectures that support both unicast and multicast traffic flows. In particular, this thesis has explored the NoC design problem in the following directions:

1. **NoC synthesis design flow**

   This thesis presents a novel NoC synthesis design flow that takes as input an application specification with performance constraints and generates an optimized NoC architecture as output that can be fed into a detailed RTL design flow. The proposed design flow incorporates floorplanning information as well as supports user-defined objectives and constraints.

2. **Novel NoC architecture synthesis algorithms to support both unicast and multicast applications**

   In general, there exists a variety of SoC applications. For many applications, support for multicast flows is necessary. Examples include the implementation of cache coherence protocols and the management of network configurations. This thesis proposes several novel NoC architecture synthesis algorithms that can support both unicast and multicast applications. These algorithms can synthesize NoC architectures that can efficiently support multicast flows without sacrificing the performance of unicast flows. To our knowledge, we are among the first to consider support for multicast applications in NoC synthesis.

3. **Extensions to support 3D NoCs and 3D applications**

   The increasing viability of three dimensional silicon integration technology has opened new opportunities for chip design innovation, including the prospect of extending emerging SoC design paradigms based on NoC interconnection architectures to 3D chip design. In this thesis, we present extensions to our NoC synthesis design flow and algorithms to support 3D-NoC synthesis. We present accurate power and delay models for 3D wiring with through-silicon vias, and we present efficient 3D-NoC synthesis algorithms that make use of these models. To our knowledge, we are among the first to consider 3D-NoC synthesis in the application-specific design domain.

4. **Extensions to support multiple traffic profiles**

   Increasingly, multiprocessor SoCs are designed to support different usage scenarios since such designs may be employed in different products or in products with different operation modes. For these embedded applications, the NoC must be designed to satisfy the communication characteristics and performance constraints of all usage scenarios considered. In this thesis, we present extensions to our NoC synthesis design flow and algorithms to support multiple traffic profiles. To our knowledge, we are among the first to consider support for multiple usage scenarios that does not assume a given or regular network topology.

5. **Deadlock-free architecture synthesis methodology**

   Finally, deadlock-free routing is a very important consideration for the correct operation of custom NoC architectures. In this thesis, several mechanisms are proposed to ensure deadlock-free routing for all flows in our synthesized NoC architectures.

1.6 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents our NoC synthesis design flow, which considers different user-defined objectives, constraints, and design parameters. The proposed design flow also incorporates floorplanning. Our problem formulation and power models are also presented in this chapter.

Chapter 3 presents the details of our proposed NoC synthesis algorithms that are based on flow-set partitioning and Steiner-trees, and Chapter 4 presents the details of an alternative NoC synthesis approach based on a rip-up and re-route concept. The experimental results presented in these chapters show that both NoC synthesis approaches can produce significantly better custom designs in terms of power, performance, as well as area when compared to regular mesh-based designs.

While both approaches are effective, we found that the rip-up and re-route approach is generally better. Therefore, we chose to extend this approach to consider two additional design dimensions. First, we describe extensions to consider 3D-NoC synthesis. These extensions are presented in Chapter 5. Second, we describe extensions to support multiple traffic profiles, which are useful for design applications that support multiple usage scenarios, each with its own traffic profile. These extensions are presented in Chapter 6.

Finally, Chapter 7 presents mechanisms for ensuring deadlock-free operation, and Chapter 8 summarizes this thesis and outlines some promising research directions.

Chapter 2

NoC Synthesis Design Flow

In this chapter, we present our NoC synthesis design flow that automates most of the complex and time-intensive design steps in NoC synthesis. It provides design support for application-specific network topologies tailored for different applications. The design flow assumes that the application has already been mapped onto cores by using pre-existing tools, and the resulting cores together with their communication requirements are taken as inputs. The ultimate goal is to design and generate a customized NoC-based communication architecture for the given application. This chapter also presents our formulation of the NoC synthesis problem as well as NoC power models for accurate power analysis.

2.1 Design Flow

Our NoC synthesis design flow is depicted in Figure 2.1. The major elements in the design flow are elaborated below.

2.1.1 Input Specification

The input specification to our design flow consists of a list of modules and their communications. As observed in recent trends, many modern SoC designs combine both hard and soft modules as well as both packet-based network communications and conventional wiring [54], as shown in Figure 2.2. Modules can correspond to a variety of different types of intellectual property (IP) cores such as embedded microprocessors, large embedded memories, digital signal processors, graphics and multimedia processors, and security

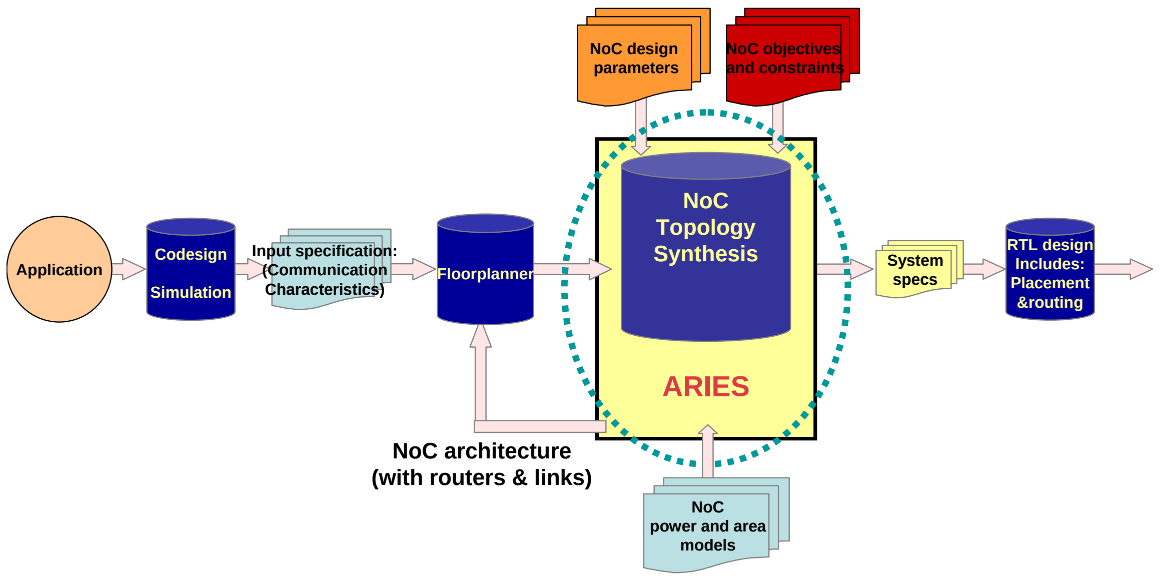

NoC design parameters
NoC objectives and constraints
Application
Codesign
Simulation
Input specification: (Communication Characteristics)
Floorplanner
NoC Topology Synthesis
ARIES
System specs
RTL design Includes: Placement &routing
NoC architecture (with routers & links)
NoC power and area models

Figure 2.1: Design flow.

encryption engines, as well as custom hardware modules. These modules can come in a variety of sizes and can be either hard or soft macros, possibly as just black boxes with area and power estimates and constraints on aspect ratios.

To facilitate modularity and interoperability of IP cores, packet-based communication with standard network interfaces is rapidly gaining adoption. Custom NoC architectures, as addressed in this chapter, are being advocated as a scalable solution to packet-based communication. For network-based communications, traffic flows with required data rates between modules are specified as part of the input specification. For our synthesis problem, we consider both unicast and *multicast* traffic flows. As discussed in [56, 65], multicast traffic flows are used in a variety of applications, and their direct support with only replication of flits at optimal bifurcation points rather than full end-to-end replication can significantly reduce network contention and resource requirements.

In general, a mixture of network-based communications and conventional wiring may be utilized as appropriate, and not all inter-module communications are necessarily over the on-chip network. For example, an embedded microprocessor may have dedicated connections to its instruction and data cache modules. Our design flow and input specification allow for both interconnection models.

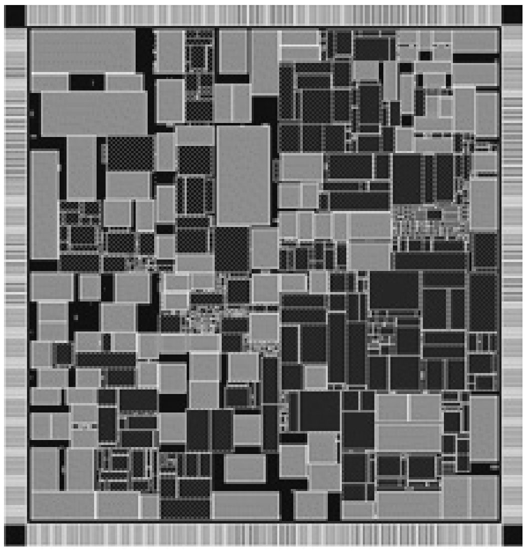


Figure 2.2: Modern SoC designs combine hard and soft modules, packet-based communications and conventional wiring-based interconnections. Source: EETimes [54].

2.1.2 Floorplanning

The floorplanning problem has been extensively studied with many mature solutions (e.g. [42, 43, 44]) These floorplanners can readily handle both hard and soft modules as well as a variety of floorplanning constraints. In general, floorplanning solutions are not restricted to a *slicing* structure, and existing methods often allow for non-slicing floorplans to achieve more efficient solutions[1]. The only requirement is that the modules are non-overlapping.

Like the NoC design flow proposed by Murali et al. [28], we have also adopted the open source floorplanner Parquet [44]. However, in the design flow proposed in [28], floorplanning is performed *after* each NoC design has been produced to evaluate detailed interconnect delays. NoC designs with different number of routers are considered. On the other hand, in our design flow, an initial floorplanning step is performed *before* NoC synthesis to obtain a placement of modules. This is important because the floorplanning of modules is often influenced by non-network-based interconnections, and the floorplan locations of modules can have a significant influence on the NoC architecture.

In particular, the input to a floorplanner like Parquet is a module netlist with

[1]A slicing floorplan is a floorplan that can be obtained by recursively cutting an enclosing rectangle by either a vertical line or a horizontal line, which restricts the possible floorplans. On the other hand, a floorplan that is not slicing is called a non-slicing floorplan. [42, 43]

weighted hyperedges. However, prior to NoC synthesis, the insertion of routers and network links has not yet been decided. To enable to the use of existing floorplanners in our design flow prior to NoC synthesis, we model traffic flows by using edge weights that are in *proportion* to the rates of traffic flows among the modules. These weighted edges are added along the side of weighted edges that represent wire connections among non-network-based modules. Since conventional floorplanners [42, 43, 44] typically support a hyperedge-based input graph model, multicast-like connectivity can be readily modeled.

With the module locations available from the initial floorplanning step, NoC synthesis can better account for wiring delays and power consumptions during the exploration of NoC architectures, including accounting for repeaters [34] on links where needed. After NoC synthesis, actual routers and links in the synthesized NoC architecture can be fed back to the floorplanner to update the floorplan, and the refined floorplanning information can be used to obtain more accurate power and area estimates. Also, NoC synthesis can be re-invoked with the refined floorplan as well. As shown experimentally in Section 3.7 and 4.3, our NoC synthesis algorithms are fast, making it feasible to iterate NoC synthesis with floorplanning.

We would like to stress that the focus of this work is on the NoC synthesis problem. We readily admit that extensions to floorplanning to better consider network-based communications is a complex problem and is a subject of separate research. Recent work has explored the development of system floorplanners that are geared towards the context of NoC synthesis [25]. Such NoC-centric floorplanners may be used in our design flow as well.

2.1.3 Networks-on-Chip Synthesis

Given floorplanning information, the NoC synthesis step then proceeds to synthesize an NoC architecture that is optimized for the given specification and floorplan.

We designed a system called ARIES to do the topology synthesis. It provides several algorithms to synthesize the best topology for application. ARIES takes the user specified objectives and constraints, the power and area models of NoC components, and the design parameters as inputs. The NoC synthesis step is to synthesize the architecture that meets all those objectives and constraints based on those parameters using those models.

Consider Figure 2.3(a) that depicts a small illustrative example. Figure 2.3(a) only shows the portion of the input specification that corresponds to the network-attached modules and their traffic flows. The nodes represent modules, edges represent traffic flows, and edge labels represent the data rate requirements for the corresponding flows. Multicast traffic flows are represented with *directed hyperedges*, which are shown graphically in Figure 2.3(a) as a *bundle* of directed edges in a *shaded* region. For example, the traffic flow from v_0 to v_1 and v_3 is a multicast flow. This graph representation is called a *communication demand graph* and is discussed in more details in Section 3.2.

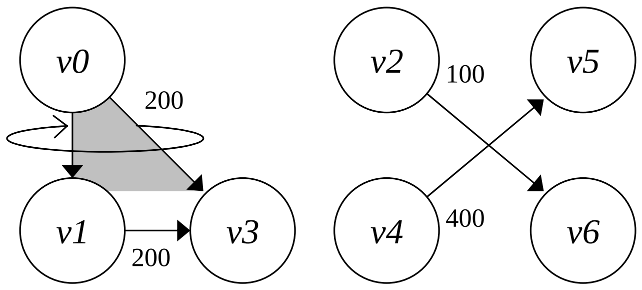
An example floorplan is shown in Figure 2.3(b). As noted earlier, modules in a design do not necessarily have to be attached to the on-chip network. Modules can also be connected by conventional wiring, as shown in the unlabeled rectangles in Figure 2.3(b). The communication demand graph with the floorplan positions annotated is illustrated in Figure 2.3(c), and Figures 2.3(d) and 2.3(e) show two example network topologies.

2.1.4 NoC Objective and Constraints

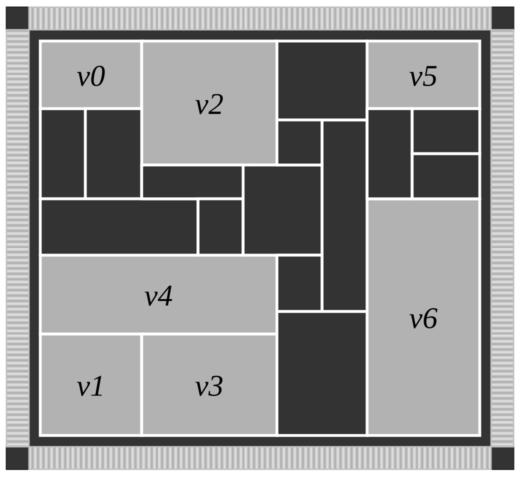
Our NoC synthesis design flow allows different user-defined objective and constraints. An important feature of our NoC synthesis algorithms, which will be detailed in the following chapters, is that they allow the *decoupling* of the evaluation cost function from the topology exploration process. As power dissipation becomes a critical issue in future IC designs due to the increased design complexity, we focus in this thesis on the problem of minimizing network power consumption under performance constraints. Other possible constraints can be design area, total wire length, or some combinations of them. Another design objective is the minimization of hop counts for data routing under power consumption constraints.

2.1.5 NoC Power and Area Estimation

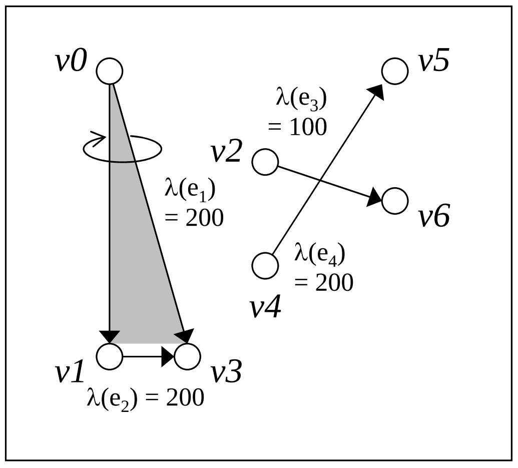
To evaluate the power and area of the synthesized NoC architecture, we use a state-of-the-art NoC power-performance simulator called Orion [31, 32] that can provide detailed power characteristics for different power components of a router for different input/output port configurations. It accurately considers leakage power as well as dynamic switching



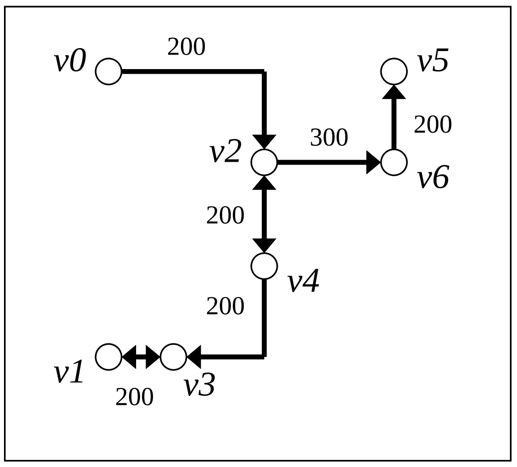
(a) Example.



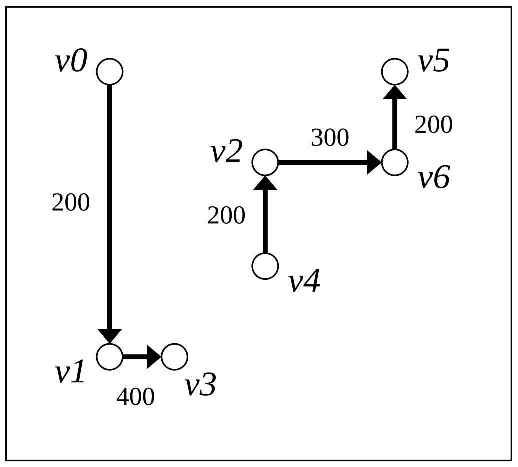
(b) Floorplan.



(c) CDG.



(d) One architecture.



(e) Alternative architecture.

Figure 2.3: Illustration of the NoC synthesis problem.

power. Orion also provides area estimates based on a state-of-the-art router microarchitecture [35, 36].

To accurately evaluate wire configurations for the network links in the synthesized architecture, we use a state-of-the-art repeated on-chip interconnect model [33, 34] to accurately determine link power consumptions, including any repeaters needed. Since floorplanning is performed in advance of NoC synthesis, wirelengths can be considered in the link power estimation. The details of these models are discussed in Section 2.2.

2.1.6 NoC Design Parameters

In addition to user-defined objectives and constraints, NoC design parameters such as the operating voltage, target clock frequency, and link widths are provided to the NoC synthesis step as well. Operating voltage and clock frequency parameters are usually dictated by the design, and link widths are often dictated by IP interface standards. However, if the design allows for different voltages or clock frequencies, or if the IP modules allow for different link widths, then NoC synthesis can be invoked to synthesize solutions for a range of design parameters specified by the user. As noted earlier, our NoC synthesis algorithms are fast, allowing for iterations with different design parameters.

2.1.7 Detailed Design

Finally, the synthesized NoC architecture with the rest of the design specification can be fed to a detailed RTL design flow where design tools like RTL optimization and detailed place and route are well established.

2.2 NoC Synthesis Problem and Formulation

2.2.1 Problem Description

The input to our NoC synthesis problem is a communication demand graph (CDG), defined as follows:

Definition 1. *A communication demand graph (CDG) is an* annotated directed hypergraph *$H(V, E, \pi, \lambda)$, where each node $v_i \in V$ corresponds to a module, and each directed hyper-*

edge $e_k = s \rightarrow D \in E$ represents a traffic flow from source $s \in V$ to one or more destinations $D = \{d_1, d_2, \ldots\}, D \subseteq V$. The position of each node v_i is given by $\pi(v_i) = (x_i, y_i)$. The data rate requirement for each communication flow e_k is given by $\lambda(e_k)$.

In general, traffic flows can either be unicast or *multicast* flows. Multicast flows are flows with $|D| > 1$. For example, in Figure 2.3(c), e_7 corresponds to a multicast flow from source v_4 to destinations v_2, v_5 and v_6.

Based on the optimization goals and cost functions specified by the user, the output of our NoC architecture synthesis problem is an optimized custom network topology with *pre-determined routes* for the specified traffic flows on the network such that the data rate requirements are satisfied. For example, Figures 2.3(d) and 2.3(e) show two different topologies for the CDG shown in Figure 2.3(c).

Figure 2.3(d) shows a network topology where all flows share a common network. In this topology, the pre-determined route for the multicast flow e_7 travels from v_4 to v_2 to first reach v_2, and then it bifurcates at v_2 to reach v_5 and v_6. Figure 2.3(e) shows an alternative topology comprising of two separate networks. In this topology, the multicast flow e_7 bifurcates in the source node to reach v_6, then it is transferred over the network link between v_4 to v_2 to reach v_2, and then bifurcates to reach v_5. Observe that in both cases, the amount of network resources consumed by routing of multicast traffic is less than what would be required if the traffic is sent to each destination as a separate unicast flow.

2.2.2 Problem Formulation

In general, the solution space of possible application-specific network architectures is quite large. Depending on the communication demand requirements of the specific application under consideration, the best network architecture may indeed be comprised of multiple networks, among each, many flows sharing the same network resources.

The goal of the proposed work in this paper is to find an optimized network topology such that the communication bandwidth requirements are satisfied and the power consumption of the network is minimized. In order to obtain the best topology solutions with minimum power consumption, accurate power models for interconnects and routers are derived. They are provided to the synthesis design flow as a library and utilized by the synthesis algorithms as evaluation criteria.

The application-specific NoC synthesis problem can be formulated as follows:

Input:

- The communication demand graph $H(V, E, \pi, \lambda)$ of the application.
- The NoC network component library $\Phi(I, J)$, where I provides the power and area models of routers with different sizes, and J provides power models of physical links with different lengths.
- The target clock frequency, which determines the delay constraint for links between routers.
- The floorplanning of the cores.

Output:

- A NoC architecture $T(R, L, C)$, where R denotes the set of routers in the synthesized architecture, L represents the set of links between routers, and a function $C : V \to R$ that represents the connectivity of a core to a router.
- A set of ordered paths P, where each $p_{ij} \in P = (r_i, r_j, \ldots, r_k)$, $r_i, \ldots, r_k \in R$, represents a route for a traffic flow $e(v_i, v_k) \in E$.

Objective:

- The minimization of power consumption for the synthesized NoC architecture.

2.3 Power Models for NoC components

In nanoscale technologies, minimizing power consumption is a very important design goal along with performance maximization. In this paper, the design goal of NoC synthesis problem is to construct an optimized interconnection architecture such that the communication requirements are satisfied and the power consumption is minimized.

The total power consumption of the communication architecture includes both leakage power and dynamic switching power of the routers and links. The dynamic switching power is a function of data rate passing through each component and the leakage power is related to the type and the characteristics of the components in the NoC architecture.

We will discuss the details of modelling these components in the following sections.

Table 2.1: Power consumption of Routers using Orion [32].

Ports (in x out)	2x2	3x2	3x3	4x3	4x4	5x4	5x5
Leakage power (W)	0.0069	0.0099	0.0133	0.0172	0.0216	0.0260	0.0319
Switching bit energy (pJ/bit)	0.3225	0.0676	0.5663	0.1080	0.8651	0.9180	1.2189

2.3.1 Modelling Routers

It is well-known that leakage power is becoming increasing dominating [32]. In the on-chip network studied in [32], leakage power represented only about 0.6% and 1.8% of the total power consumption at 180nm and 100nm, respectively, but leakage power increased to 25% at 70nm. High-performance microprocessor studies show even a much larger leakage power component [30]. Therefore, it is important to properly account for leakage power when adding routers and channels to the synthesized architecture. However, when considering leakage power, the cost function may need to account for possibly discrete cost increments of links and routers whereas dynamic switching power may be best modelled as a function of cumulative data rates. This non-linear characteristic of the power consumption of the NoC makes it hard to be accurately modelled using MILP or LP formulations.

To evaluate the power of the routers in the synthesized NoC architecture, We use a state-of-the-art NoC power-performance simulator called Orion [31, 32] that can provide detailed power characteristics for different power components of a router for different input/output port configurations. It accurately considers leakage power as well as dynamic switching power. Orion also provides area estimates based on a state-of-the-art router microarchitecture [35, 36]. The power cost of a router is modelled by its major building blocks, namely the input buffers, crossbars, and arbiters. The power per bit values are used as the basis for the entire router dynamic power estimation under different configurations. The leakage power and switching bit energy of some example router configurations with different number of ports in 70nm technology are showed in Table 2.1.

2.3.2 Modelling Interconnects

In the NoC architecture, interconnects can be modelled as distributed RC wires. As discussed in Section 2.2.2, the target clock frequency is provided to our NoC synthesis design flow as a design parameter. Depending on the network topology, long intercon-

Table 2.2: Interconnect Parameters

Electrical		Physical	
$\rho = 2.53\mu\Omega \cdot cm$	k_{ILD}= 2.7	w= 500nm	s= 500nm
r_h= 46Ω/mm	c_h = 192.5fF/mm	t = 1100nm	h= 800nm

Table 2.3: Power consumption of interconnects.

Wire length (mm)	1	4	8	12	16
Leakage power (W)	0.000496	0.001984	0.003968	0.005952	0.007936
Switching bit energy (pJ/bit)	0.6	2.4	4.8	7.2	9.6

nects may be required to implement network links between routers, which may have wire delays that are larger than the target clock frequency. To achieve the target frequency, repeaters may need to be inserted. Thus we use the state-of-art repeated on-chip interconnect model [33, 34] where the interconnect is evenly divided into k segments with repeaters inserted between them that are s times as large as a minimum-sized repeater. The delay and power consumption per bit of this interconnect can be modelled using the Elmore model, as in [33, 34]. When minimizing power consumption is the objective, the optimum size s_{opt} and number k_{opt} of repeaters that minimize power consumption while satisfying the delay constraint can be determined for the interconnect using the method proposed in [33].

In our experiments, the physical and electrical parameters in $70nm$ technology are used and are listed in Table 2.2. The wires are implemented on the global metal layers and their parameters are extracted from ITRS [2].

In our NoC synthesis design flow, we use the above interconnect model to evaluate optimum power consumption of interconnects with different wire lengths under the given design frequency and delay constraints. These results are provided to the design flow in the form of a library. Since the floorplanning is performed in advance of NoC synthesis, wirelength is known for each on-chip interconnect when evaluating the power consumption.

Table 2.3 lists the static power and switching bit energy parameters of some example interconnects with different wirelengths in 70nm technology under 1GHz frequency constraints.

Acknowledgement

This chapter is in part a reprint of the materials in the following two papers: Shan Yan, Bill Lin, “Custom Networks-on-Chip Architectures with Multicast Routing”, *IEEE Transactions on VLSI Systems*, volume: 17, issue: 3, on pages: 342-355, March 2009; Shan Yan, Bill Lin, “Joint Multicast Routing and Network Design Optimization for Networks-on-Chip”, *IET Computers and Digital Techniques*, accepted for publication, 2009. The dissertation author was the primary author of both papers.

Chapter 3

Design Custom Topologies Based on Flow-Set Partitions and Steiner Trees

3.1 Overview

In this chapter, we describe our NoC synthesis algorithms based on the set-partition of flows and Steiner Tree. Our problem formulation is based on the decomposition of the problem into the inter-related steps of finding good flow partitions, deriving a good physical network topology for each group in the partition, and providing an optimized network implementation for the derived topologies. Our solutions may be comprised of multiple custom networks, each interconnecting a subset of communicating modules. In particular, we propose four algorithms for systematically examining different set partitions of communication flows. The first two are heuristic algorithms called CLUSTER and DECOMPOSE, and the last two are perturbation-based probabilistic algorithms called PERTURB (perturbation-based flow partitioning) and R-PERTURB (reduced perturbation-based flow partitioning).

For each set partition considered, we use well-developed Rectilinear-Steiner-Tree (RST) algorithms [39, 40, 41] to generate a physical network topology for each group in the set partition. Though the RST problem is in itself NP-hard, well-developed fast RST algorithms are available that can be effectively used, as indicated by the run-times presented in Section 3.7. For each RST derived, the routes for the corresponding flows and the bandwidth requirements for the corresponding network links are determined. Our formulation

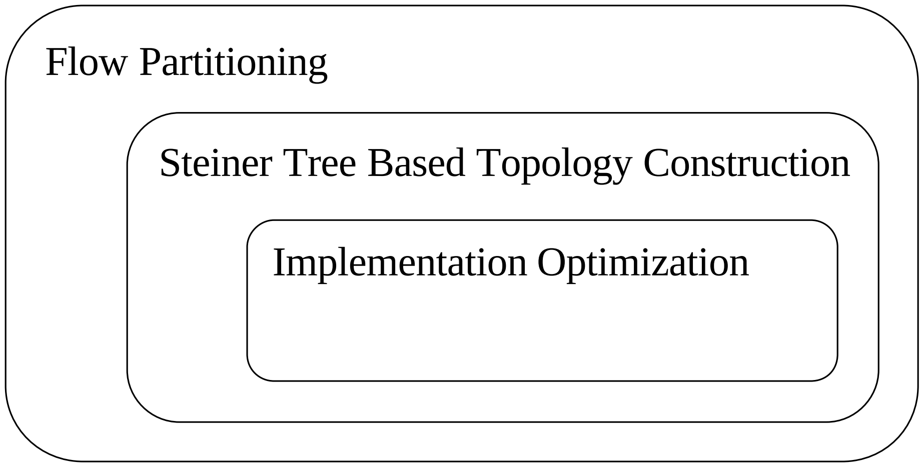

Flow Partitioning
Steiner Tree Based Topology Construction
Implementation Optimization

Figure 3.1: Formulation of synthesis problem.

supports a decoupling of the evaluation cost function from the exploration process, which enables the flexible incorporation of different design objectives and constraints. Although we use Steiner trees to generate a physical network topology for each group in the set partition, the final NoC architecture synthesized is not necessarily limited to just trees as RST implementations of different groups may be connected to each other to form non-tree structures.

The rest of this chapter is organized as follows. Section 3.2 presents the problem description and our formulation. Sections 3.3 and 3.4 describe the CLUSTER and DECOMPOSE, respectively. Section 3.5 describes PERTURB and R-PERTURB. Section 3.6 describes router merging algorithm. Finally, experimental results are presented in Section 3.7.

3.2 Problem Formulation

The synthesis formulation is depicted in Figure 3.1 and each element is elaborated below.

3.2.1 Flow-Set Partitioning

Flow partitioning is performed in the outer loop of our synthesis formulation to explore different partitioning of flows to separate sub-networks. In general, the solution space of distinct set partitions of n flows, commonly known as the n^{th} Bell number, is known to grow $\Theta(n \log n)^n$ [38]. B_n grows rapidly, with e.g. B_{10}, B_{11}, and B_{12} equal to 115975, 678570, and 4213597, respectively, and so on. The goal of the heuristic algorithms CLUSTER and DECOMPOSE presented in Sections 3.3 and 3.4 is to significantly reduce the number of flow partitions that we consider in a systematic manner. In Section 3.5, we also present two simulated annealing based flow partitioning algorithms called perturbation-based flow partitioning (PERTURB) and reduced perturbation-based flow partitioning (R-PERTURB).

3.2.2 Steiner Tree Based Topology Construction

For each flow partition considered, physical network topologies must be decided for carrying the traffic flows. In current process technologies, layout rules for implementing wires dictate physical topologies where the network links run horizontally or vertically. Thus, the problem is similar to Rectilinear Steiner Tree (RST) problem that has been extensively studied for the conventional VLSI routing problem. Given a set of nodes, the RST problem is to find a network with the shortest edge lengths using horizontal and vertical edges such that all nodes are interconnected. The RST problem is well-studied with very fast implementations available [39, 40]. Figures 2.3(d) and 2.3(e) show possible RSTs for the set partitions $\{\{1, 2, 3, 4\}\}$ and $\{\{1, 2\}, \{3, 4\}\}$, respectively. We use an RST solver in the inner loop of flow partitioning to generate topologies for the set partitions considered. RSTs are only re-evaluated for the groups in the set partition that changed at each step, and previously computed RSTs can be cached.

Since most existing Steiner tree algorithms are based on a hypergraph model, they match closely to our topology construction problem for multicast traffic flows. In addition, emerging CMOS technologies are providing an increasing number of metal layers for implementing network links, facilitating "over-the-module" routing of network links. At 65nm, as much as 11 copper metal layers has been demonstrated [55]. However, the use of some hard IP macros may prohibitive routing over them. Fortunately, our Steiner-tree based topology construction formulation allows for the use of available obstacle-avoiding RST algorithms [41] to accommodate this type of constraints.

After a physical topology is generated for each group in a set partition, the routes for the corresponding flows and the bandwidth requirements for the corresponding network links can be readily derived. The routes for the flows follow directly from the tree structure of the RST solution. For example, in Figure 2.3(d), the route for the unicast flow e_4 travels from v_4 through v_2 and v_6 to reach v_5. For the multicast flow e_1, a flit is first sent from v_0 through v_2 and v_4 to v_3. The flit is then copied and forwarded to v_1 over the network link from v_3 to v_1. Correspondingly, the cumulative bandwidth requirements for each network link can also be readily derived from the RST solution, as for example shown in Figures 2.3(d) and 2.3(e).

3.2.3 Implementation Optimization

Given the RST-based topologies generated, network implementations are next derived. In particular, a separate network implementation is initially generated for each RST. Routers are allocated at junctions where either flows from multiple links must be multiplexed to the same outgoing link or flows from the same link must be de-multiplexed to multiple outgoing links, and network links are allocated to connect routers and network interfaces. To segment long links, repeaters are inserted as required [34]. Although the RST problem generates a graph with undirected edges, network links and router ports are only allocated in the direction of traffic flows. After an initial network implementation is generated for each RST, the separate networks are combined together to form a complete NoC architecture.

To improve upon the initial NoC architecture generated, a greedy router merging procedure is performed to further optimize the implementation and reduce cost. The detail of the router merging algorithm is discussed in Section 3.6. It works iteratively by considering all possible mergings of two routers connected with each other in each iteration and keeps merging routers until no improvement can be made further.

3.3 CLUSTER

In this section, we present an algorithm called CLUSTER that reduces the number of set partitions considered from $\Theta(n \log n)^n$ to $\Theta(n^3)$, which is a significantly smaller subset of set partitions. The details of the algorithm is shown in Algorithm 1. The CLUS-

TER algorithm takes a communication demand graph and an evaluation cost function as input and generates an optimized network architecture implementation details as output. As discussed before, our synthesis formulation provides a decoupling of the evaluation cost function from the exploration process. In particular, the CLUSTER algorithm starts by implementing each edge in the communication demand graph separately. The solution for each single edge is a simple RST connecting two terminals. It sets these single edges as the initial set partition, denoted as $P^0 = \{\{e_1\}, \{e_2\}, \ldots, \{e_n\}\}$, as shown in lines 2-5.

Then, in lines 7-18, at each iteration, the algorithm systematically generates new candidate set partitions starting from the set partition chosen from the previous iteration. In particular, in the first iteration, the algorithm starts with the initial set partition $P^0 = \{\{e_1\}, \{e_2\}, \ldots, \{e_n\}\}$ with n groups, each group containing exactly one edge. Then, in lines 8-13, the algorithm generates new candidate set partitions from P^0 by considering all pairwise mergings of groups in P^0. The groups are denoted as g_u and g_v in the pseudo-code. For each pairwise merging considered, an RST solver is called to generate a physical network topology for the merged set of flows, and the cost of this network is calculated using the specified evaluation function C. We do not need to solve an RST problem for the entire set of flows, just the subset of flows in the merged groups is considered. We then, in line 12, compute the total cost of the resulting set partition by summarizing the cost of implementing all the other sets using their own networks. In lines 15-16, we select the merging that achieves the best cost in this iteration and choose it as P^1. In general, we start from the chosen set partition, P^t, from the iteration to generate pairwise mergings of groups from P^t, and the best merging is selected as the new chosen set partition P^{t+1}. At each iteration, the number of groups that need to be considered is reduced by 1, but the size of groups will become increasingly larger. Finally, in the last iteration, we only need to consider the mergings of two groups.

At each iteration in lines 7-18, we maintain the chosen set partition and the associated cost calculations for that iteration. Then, in the end of the algorithm, lines 23-24, we choose the set partition with the minimum cost. Since at each iteration t, there can be at most $(n-t)(n-t-1)/2$ possible pairwise group mergings, and there are $(n-1)$ iterations, the number of set partitions considered in the CLUSTER algorithm is $\Theta(n^3)$.

Algorithm 1 CLUSTER $(G(V, E, \pi, \lambda), C, T)$

Input: $G(V, E, \pi, \lambda)$: communication demand graph

C: specified evaluation function for implementation cost

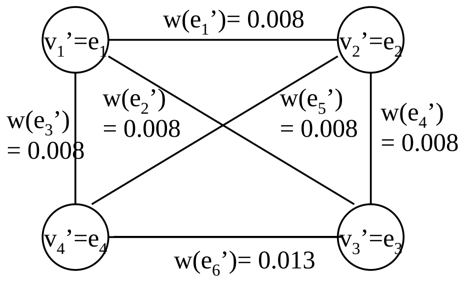
Output: T: synthesized network architecture

```
 1: initialize $P^0 = \varnothing$
 2: **for all** $e_k \in E$ **do**
 3:     $P^0 = P^0 \cup \{e_k\}$
 4:     $cost(\{e_k\})$ = EvaluateCost($T(\{e_k\}), C$)
 5: **end for**
 6: $t = 0$
 7: **while** $|P^t| > 1$ **do**
 8:     **for all** $g_u, g_v \in P^t$ **do**
 9:         $g_{uv}$ = $g_u \cup g_v$
10:         $T(g_{uv})$ = SolveRST($g_{uv}$)
11:         $cost(g_{uv})$ = EvaluateCost($T(g_{uv}), C$)
12:         $\beta(g_u, g_v) = cost(g_{uv}) + \sum_{g_i \in P^t, g_i \neq g_u, g_v} cost(g_i)$
13:     **end for**
14:     $(u, v) = \arg\min_{g_u, g_v \in P^t} \beta(g_u, g_v)$
15:     $P^{t+1} = P^t \backslash \{g_u, g_v\}$
16:     $P^{t+1} = P^{t+1} \cup \{g_u \cup g_v\}$
17:     $t = t + 1$
18: **end while**
19: **for all** $t \in [0, n-1]$ **do**
20:     $c(P^t) = \sum_{g_u \in P^t} cost(g_u)$
21:     $soln[P^t] = \bigcup_{g_u \in P^t} T(g_u)$
22: **end for**
23: $t = \arg\min_{t \in [0, n-1]} c(P^t)$
24: $T = soln[P^t]$
25: **return** $T$
```

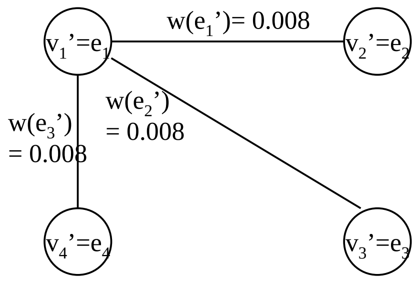
3.4 DECOMPOSE

The DECOMPOSE algorithm described in this section reduces the number of set partitions considered from $\Theta(n \log n)^n$ to $\Theta(n^2)$. The details of the algorithm is shown in Algorithm 2. This algorithm works in the opposite direction as CLUSTER when generating candidate set partitions and the corresponding RST topologies. It starts by considering all communication demands as a single cluster. In each iteration, the algorithm considers different ways of breaking up an existing group in the set partition chosen from the previous iteration into two smaller ones. Then, the differential cost of splitting a group is evaluated by generating an RST for each sub-group and evaluating their costs using the specified evaluation function. To facilitate this decomposition process, two important graphs are used in DECOMPOSE: Affinity Graph (AG) and its Minimum Spanning Tree (MST). The affinity graph A is built by associating each flow in the communication demand graph to a vertex in the affinity graph. An edge is added between each pair of the vertices in the affinity graph to form a complete graph. A weight is attached to each edge $e' = (v'_i, v'_j)$ and is calculated as $w(e') = cost(\{e_i, e_j\}) + \sum_{e_k \in E, e_k \neq e_i, e_j} cost(\{e_k\})$, where e_i is the flow in the communication demand graph associated with v'_i in the affinity graph. $cost(\{e_k\})$ is calculated by calling on the evaluation function to evaluate the cost of implementing $\{e_k\}$ separately and $cost(\{e_i, e_j\})$ is calculated by evaluating the cost of implementing a generated RST topology for $\{e_i, e_j\}$ together. The weights of the edges in the affinity graph reflect the benefits of implementing flows represented by vertices in the affinity graph together using shared resources. The affinity graph used here is based on the similarity graph model proposed in [24], except that RSTs are used to determine the affinities. In particular, in the affinity graph, the smaller the weight, the less the resulting total cost of clustering the two flows connected by that edge. The motivation is to only cluster flows that are connected by small weighted edges so that the total implementation cost is minimized. Then the minimum spanning tree M of A that contains the minimum number of minimal weighted edges connecting all the vertices in A is derived. The affinity graph and its MST of the example in Figure 2.3 are shown in Figure 3.2. The cost considered is the total power consumption based on the 70nm technology power estimations shown in Table 2.1.

Recall that the vertices in the spanning tree M corresponds to *flows* in the communication demand graph G. Since M is initially a spanning tree, it interconnects all



(a) Affinity graph.



(b) MST of affinity graph.

Figure 3.2: Affinity graph and MST for example shown in Figure 2.3.

vertices, which is interpreted as having all *flows* in a single cluster. During the course of the DECOMPOSE algorithm, we will selectively remove edges from M to create *disjoint set of vertices*, which will correspond to *disjoint sets of flows into groups*, thus forming a particular set partition.

In each iteration shown in lines 5-9, the algorithm systematically generates new candidate set partitions starting from the set partition chosen from the previous iteration. Inside the while loop, new set partitions are generated by temporarily removing one edge from M. This is achieved by calling the routine SelectEdgeToDelete(M). With an edge removed, the corresponding group is split into two sub-groups. We evaluate the cost of this splitting by solving an RST problem for each sub-group and calling on the evaluation function to compute the new costs. In the first iteration of the algorithm, the spanning tree M has $(n-1)$ edges. Thus, $(n-1)$ new candidate set partitions will be generated. The set partition with the best cost will be chosen as the set partition for the current iteration. This set partition, and the corresponding modified M, will be used as the starting point for the next iteration. At iteration t, M will have $(n-t-1)$ remaining edges. Therefore, $(n-t-1)$ candidate set partitions will be generated and considered. The algorithm ends when all flows in the problem have been split into their own individual groups. Then, at the end of the algorithm, at line 10, we choose the set partition with the minimum cost among the set partitions chosen from all iterations. Since at each iteration t, there can be at most $(n-t-1)$ candidate set partitions, the number of set partitions considered in the DECOMPOSE algorithm is $\Theta(n^2)$, which is again considerably smaller than $\Theta(n \log n)^n$.

Algorithm 2 DECOMPOSE $(G(V, E, \pi, \lambda), C, T)$

```
Input: $G(V, E, \pi, \lambda)$: communication demand graph
    $C$: specified evaluation function for implementation cost
Output: $T$: synthesized network architecture
 1: $A$ = GenerateAffinityGraph($G$)
 2: $M$ = GenerateMinSpanningTree($A$)
 3: $t = 0$
 4: $n = |E|$
 5: while $|M| < n$ do
 6:     $(e, soln[t], cost[t])$ = SelectEdgeToDelete($M$)
 7:     remove $e$ from $M$
 8:     $t = t + 1$
 9: end while
10: $T = soln[\arg\min_t cost[t]]$
11: return $T$
SelectEdgeToDelete($M$)
 1: for all $e_i \in M$ do
 2:     temporarily remove $e_i$ from $M$
 3:     $components(M)$ = CalculateConnectedComponents($M$)
 4:     for all $g_i \in components(M)$ do
 5:         $T(g_i)$ = SolveRST($g_i$)
 6:         $soln[e_i] = soln[e_i] \cup T(g_i)$
 7:         $cost[e_i] = cost[e_i]$ + EvaluateCost($T(g_i), G, C$)
 8:     end for
 9:     add $e_i$ back to $M$
10: end for
11: $e = \arg\min_{e_i \in M} cost[e_i]$
12: return $(e, soln[e], cost[e])$
```

3.5 Perturbation-Based Flow Partitioning

In this section, we present two alternative flow partitioning algorithms called PERTURB (perturbation-based flow partitioning) and R-PERTURB (reduced perturbation-based flow partitioning). These algorithms are based on the use of simulated annealing [45], which has been successfully employed in numerous global optimization problems. Sections 3.5.2 to 3.5.4 describe PERTURB, and Section 3.5.5 describes a more efficient version called R-PERTURB that considers a reduced state space.

3.5.1 Overview of SA

Simulated annealing (SA) is a generic perturbation-based probabilistic global optimization algorithm [45]. Each step of the SA algorithm replaces the current solution by a random perturbation, chosen with a probability that depends on the change in cost and a global temperature parameter T. We use the transition probability from [45], which is defined as 1 if $\Delta = c' - c$ is negative, where c' and c are the costs of the new state and the previous state, respectively (i.e., reduced cost moves are always accepted). Otherwise, the acceptance probability is $e^{-\Delta/T}$. The allowance for "increased cost" moves prevents SA from becoming stuck at local minima.

In order to apply the SA method to a specific problem, three questions must be resolved: How should the solution space be represented? How should new candidates be generated? And what cost function should be used to evaluate each candidate? Each of these questions is addressed below.

3.5.2 Representing the State Space

SA based optimization requires a representation of the state space that can be searched to find optimal solutions. PERTURB takes a communication demand graph and an evaluation function as inputs and generates an optimized network architecture as output. Given a communication demand graph that specifies all the communication demand flows in the application, the goal is to explore the state space of different possible set partitions of the flows, implement each set partition as groups of Rectilinear Steiner Trees (RSTs), and evaluate the cost of each implementation. During the annealing process, best solution seen is maintained. The solution space considered by PERTURB is the set of $\Theta(n \log n)^n$ set partitions, although only a subset of candidates are evaluated during the simulated annealing process.

3.5.3 Generating Candidate Solutions

PERTURB works by randomly moving to a neighbor state from the current state in each step and evaluating the benefit of the move. The candidate set partition is generated from the current set partition by the function GenerateNewSetPartition(), as shown in Algorithm 3. It takes the current set partition configuration and communication demand graph as input. Two flows are randomly selected from the communication demand graph and the sets that they belong to in the current partition are found as S_0 and S_1. If the two flows are in different sets of the current set partition, the corresponding two sets are merged into one set in the new partition; otherwise, the set is randomly split into ${S_0}^{new}$ and ${S_1}^{new}$ by randomly assigning one flow in ${S_0}^{new}$ and the other in ${S_1}^{new}$ and randomly assigning remaining flows in the original set to ${S_0}^{new}$ and ${S_1}^{new}$. The new set(s) generated together with the remaining sets in the original partition form the new set partition and are returned. This method of generating new set partitions allow use to traverse the $\Theta(n \log n)^n$ state space of possible set partitions.

3.5.4 Incremental Cost Evaluation

For each set partition of the flows generated in each step of PERTURB, an RST solver is called to generate a physical network topology for the configuration. The flows in each set in the set partition are considered as a cluster and an RST instance is solved as the physical network topology of this cluster. Note that for each new set partition generated from the old one, at most two clusters can change, by either merging two clusters into one or splitting one cluster into two. All others remain unchanged. So we only incrementally generate and solve the RST instances for the clusters merged or the cluster split. The derived RSTs are then implemented and evaluated as discussed in Section 3.2.3.

Algorithm 3 GenerateNewSetPartition($P, G(V, E, \pi, \lambda)$)

```
Input: $P$: input set partition
    $G(V, E, \pi, \lambda)$: communication demand graph
Output: $P'$: new generated set partition
 1: randomly select 2 flows $e_0, e_1 \in E$, $e_0 \neq e_1$
 2: $S_0 = set(e_0), S_1 = set(e_1)$
 3: if $S_0 \neq S_1$ then
 4:     $P' = P \backslash \{S_0, S_1\}$
 5:     $P' = P' \cup \{S_0 \cup S_1\}$
 6: else
 7:     ${S_0}^{new} = \{e_0\}$
 8:     ${S_1}^{new} = \{e_1\}$
 9:     for all $e_i \in S_0, e_i \neq e_0, e_1$ do
10:         if random(0, 1) $>$ 0.5 then
11:             ${S_0}^{new} = {S_0}^{new} \cup \{e_i\}$
12:         else
13:             ${S_1}^{new} = {S_1}^{new} \cup \{e_i\}$
14:         end if
15:     end for
16:     $P' = P \backslash \{S_0\}$
17:     $P' = P' \cup \{{S_0}^{new}, {S_1}^{new}\}$
18: end if
19: return $P'$
```

3.5.5 Reducing the State Space

The PERTURB algorithm presented above explores a state space that comprises of all possible set partitions. However, the large state space that it considers can lead to long run times. In this section, we present another SA-based algorithm called Reduced-PERTURB (R-PERTURB) that reduces the size of the state space from $\Theta(n \log n)^n$ set partitions to $\Theta(2^n)$. In practice, R-PERTURB can significantly reduce run times while still achieving good results as compared to PERTURB.

R-PERTURB works in the similar way as PERTURB except it uses a different neighbor selection method to generate candidate solutions. In order to efficiently reduce the number of set partitions considered without excluding potentially good candidates, R-PERTURB again makes use of affinity graphs and minimum spanning trees, as used in the DECOMPOSE algorithm. Before the SA process starts, an affinity graph A and its minimum spanning tree M are first derived from the communication demand graph, as described in Section 3.4. Referring again to Figure 3.2, it shows the affinity graph and its minimum spanning tree for the example shown in Figure 2.3.

For the initial minimum spanning tree generated, the edges in M are numbered and saved in N. At each iteration, a new neighboring set partition is generated from the current set partition by using the function GenerateNewSetPartition() shown in Algorithm 4. It works by randomly selecting an edge in the initial minimum spanning tree saved in N. If the selected edge is in the current M, it is removed from M; otherwise, it is added to M. By adding or removing edges to/from M, the set partition represented by M changes. The new set partition is saved to P' in the function GetDisjointSets() by finding all the disjoint sets of vertices in M, which correspond to disjoint sets of flows.

For a communication demand graph containing n flows, the number of vertices in the affinity graph A is n, and the number of edges in the minimum spanning tree is $n-1$. By considering adding or removing edges in the spanning tree to generate new set partitions, R-PERTURB considers a reduced state space of $\Theta(2^n)$ possible set partitions, which is much smaller than $\Theta(n \log n)^n$. Similar to PERTURB, only a subset of candidates in the reduced state space are evaluated during the simulated annealing process.

Algorithm 4 GenerateNewSetPartition(M, N)

```
Input: $M$: modified MST of current set partition.
    $N$: original MST edges array
Output: $P'$: new generated set partition
1: randomly select 1 edge $e_i$ from $N$
2: if $e_i \in M$ then
3:     remove $e_i$ from $M$
4: else
5:     add $e_i$ to $M$
6: end if
7: $P'$ = GetDisjointSets($M$)
8: return $P'$
```

3.6 Router Merging

After the physical network topology has been generated using the above algorithms, a router merging step is used to further optimize the topology to reduce the power consumption cost. The router merging step was first proposed by Srinivasan in [27]. Their router merging was based on the distance between routers. However, in our work, we propose a new router merging algorithm for reducing the power consumption of the network and improving the performance.

As has been observed, routers that connect with each other can be merged to eliminate router ports and links and thus possibly the corresponding costs. Routers that connect to the same common routers can also be merged to reduce ports and costs. We propose a greedy router merging algorithm, which is shown in Algorithm 5. The algorithm works iteratively by considering all possible mergings of two routers connected with each other. In each iteration, each router's adjacent routers list are constructed and sorted by the distance between them in increasing order. They are possible candidate mergings. Then the routers are considered to merge in the decreasing order of the number of neighbors they have. For each candidate merging, if the topology from the merging result is valid, the total power consumption of the resulting topology after merging is evaluated using the power models. Routers are merged if they have not merged in this iteration and the cost is improving. After all routers are considered in the current iteration, they are updated by replacing the routers merged with the new one generated. Those routers are reconsidered in the next iteration. The algorithm keeps merging routers until no improvement can be made further. After router merging, the optimized topology is generated and the routing paths of all flows are updated. Since the router merging will always reduce the number of routers in the topology, it will not increase the hop counts for all the flows thus will not worsen the performance of the application. The topology generated after router merging represents the best solution with the minimum power consumption. It is returned as the final solution for our NoC synthesis algorithm.

3.7 Experimental Results

3.7.1 Experimental Setup

We have implemented our four proposed algorithms CLUSTER, DECOMPOSE, PERTURB and R-PERTURB in C++. In our implementation, we incorporated a fast public domain Rectilinear Steiner Tree solver called GeoSteiner4.0 [39, 40] to generate the physical network topologies in the inner loop of the four algorithms. The proposed router merging algorithm has been integrated into the four algorithms as well to improve the solutions generated. As discussed in the design flow outlined in Chapter 2, we use Parquet [44] for the floorplanning step.

To evaluate our proposed synthesis algorithms, two groups of experiments were used. The first group of experiments was used to evaluate the performance of our algorithms on applications with only unicast flows. The second group of experiments was used to evaluate the performance of our algorithms on benchmarks with multicast traffic flows.

For the first group of experiments, three sets of benchmarks were used to evaluate the proposed algorithms. The first set of benchmarks consists of four different video processing applications obtained from [29], including a Video Object Plane Decoder (VOPD), an MPEG4 decoder, a Picture-In-Picture (PIP) application, and a Multi-Window Display (MWD) application. The next set of benchmarks were obtained from [20] and [25]. They correspond to different encoder/decoder combinations of a H.263 video codec, a MP3 audio codec, and a generic MultiMedia System (MMS). Finally, to generate larger benchmark instances, we generated synthetic benchmarks from the above video applications.

Algorithm 5 ROUTER-MERGING(R, T)

```
Input: $R$: routers list $T$: network topology
Output: $R'$: new routers list; $T'$: new network topology
 1: $done = 0$
 2: while $done = 0$ do
 3:     for all $r_i \in R$ do
 4:         $adj(r_i)$ = generate all 1-hop adjacent router list and sort it by their distance to $r_i$
            in increasing order
 5:     end for
 6:     sort routers in $R$ by their number of adjacent routers in decreasing order
 7:     for all $r_i \in R$ in this order do
 8:         for all $r_j \in adj(r_i)$ do
 9:             if neither $r_i$ nor $r_j$ is merged in this round then
10:                 evaluate the total power consumption cost of merging $r_i$ and $r_j$
11:                 merge $r_i$, $r_j$ to $r'$ if merging is valid and total power consumption is improv-
                    ing
12:                 delete $r_i$, $r_j$ from $R$ and add $r'$ to $R$
13:             end if
14:         end for
15:     end for
16:     if no merging is done in this iteration then
17:         $done = 1$
18:     end if
19: end while
20: update topology $T'$ and routing path for all flows
21: return $R' = R, T'$
```

For the second group of experiments, in the absence of published benchmarks with multicast traffic, we generated a set of synthetic benchmarks. In particular, we used the NoC-centric bandwidth-version of Rent's rule proposed in [46] to generate these benchmarks. The details of this benchmark generation process is described in Section 3.7.3.

All experimental results were obtained on a 1.5 GHz Intel P4 processor machine with 512 MB of memory running Linux.

3.7.2 Results for Unicast Applications

Method of evaluation

In all our experiments, we aim to evaluate the performance of the four proposed algorithms CLUSTER, DECOMPOSE, PERTURB, and R-PERTURB on all benchmarks with the objective of minimizing the total power consumption of the synthesized NoC architectures. The total power consumption includes both the leakage power and the dynamic switching power of all network components. As discussed in Chapter 2, we use a power-performance simulator called Orion [31, 32] to estimate the power consumptions of router configurations generated. We applied the design parameters of 1 GHz clock frequency, 4-flit buffers, and 128-bit flits. For the link power parameters, we use the state-of-art on-chip repeated interconnect model [33, 34] to evaluate the optimum powers for links with different lengths under the given delay constraint of 1ns. Both routers and links are evaluated using the 70nm technology and they are provided in a library.

For evaluation, fair direct comparison with previously published NoC synthesis results is difficult in part because of vast differences in the power parameters assumed[1]. To evaluate the effectiveness of our algorithms, we have designed two sets of experiments. In the first set of experiments, we generated a full mesh implementation for each benchmark for comparison. For the positioning of modules on the mesh implementation, we again used Parquet [44]. In a full mesh implementation, each module is connected to a router with 5 input/output ports. Packets are routed using XY routing over the mesh from source to destination. We also generated a variant of the basic mesh topology called optimized mesh (opt-mesh) by eliminating router ports and links that are not used by the traffic flows.

[1]We use the Orion simulator to estimate power consumption [31, 32]. The power estimates are consistent with another published power-optimized NoC implementation described in [37]. The power estimates are on the same order of magnitude for the same router configuration in the same technology.

Table 3.1: NoC power comparisons on unicast applications.

				CLUSTER				DECOMPOSE				R-PERTURB				PERTURB				mesh	opt-mesh
Appli.	Label	|V|	|E|	Power (W)	Time (sec)	Improv. CL/ mesh	Improv. CL/opt	Power (W)	Time (sec)	Improv. DE/ mesh	Improv. DE/opt	Power (W)	Time (sec)	Improv. R-P/ Mesh	Improv. R-P/ Opt	Power (W)	Time (sec)	Improv. PERT./ Mesh	Improv. PERT./ opt	Power (W)	Power (W)
VOPD	U1	12	14	0.042	1.35	6.42	3.60	0.042	0.34	6.42	3.60	0.042	8	6.42	3.60	0.042	210	6.42	3.60	0.272	0.152
MPEG4	U2	12	13	0.039	1.09	7.07	2.35	0.040	0.30	6.84	2.27	0.040	10	6.89	2.29	0.039	159	7.08	2.35	0.276	0.092
PIP	U3	8	8	0.021	0.24	8.65	2.93	0.021	0.09	8.65	2.93	0.021	3	8.65	2.93	0.021	38	8.65	2.93	0.178	0.060
MWD	U4	12	13	0.028	1.57	9.24	4.31	0.031	0.32	8.44	3.94	0.028	8	9.24	4.31	0.026	180	9.91	4.62	0.260	0.121
H263	U5	7	10	0.037	0.37	4.18	2.17	0.036	0.14	4.31	2.24	0.036	7	4.31	2.24	0.035	75	4.38	2.27	0.155	0.080
G5	U6	12	12	0.028	0.82	9.11	3.76	0.031	0.23	8.29	3.42	0.031	7	8.35	3.45	0.028	126	9.11	3.76	0.258	0.106
mp3enc	U7	7	8	0.023	0.09	4.28	2.16	0.023	0.05	4.28	2.16	0.023	1	4.28	2.16	0.023	18	4.28	2.16	0.100	0.050
mp3dec	U8	6	6	0.016	0.11	6.02	2.60	0.016	0.04	6.02	2.60	0.016	1	6.02	2.60	0.016	13	6.03	2.60	0.099	0.043
h263dec	U9	7	8	0.026	0.19	6.77	2.47	0.026	0.07	6.77	2.47	0.026	4	6.77	2.47	0.026	34	6.77	2.47	0.178	0.065
MMEnc	U10	14	19	0.068	3.89	5.50	1.91	0.073	0.77	5.12	1.78	0.068	17	5.50	1.91	0.068	400	5.50	1.91	0.376	0.131
MMDec	U11	11	14	0.054	1.88	4.74	2.24	0.054	0.32	4.74	2.24	0.054	14	4.74	2.24	0.049	186	5.22	2.47	0.257	0.122
MMS	U12	25	33	0.123	32.07	5.21	2.11	0.132	3.96	4.85	1.96	0.121	47	5.29	2.14	0.126	1735	5.08	2.06	0.642	0.260
V+M	U13	24	27	0.086	15.47	7.39	2.35	0.090	1.30	7.04	2.24	0.082	32	7.73	2.46	0.081	1100	7.81	2.49	0.633	0.202
M+P	U14	20	21	0.053	7.45	9.29	2.89	0.053	0.96	9.29	2.89	0.053	11	9.29	2.89	0.053	620	9.29	2.89	0.495	0.154
V+M+M	U15	36	40	0.120	73.02	8.37	2.49	0.124	7.54	8.13	2.42	0.122	54	8.26	2.45	0.119	3248	8.47	2.52	1.008	0.300
4in1	U16	44	48	0.139	130.50	10.25	2.47	0.142	12.68	10.02	2.42	0.141	80	10.10	2.44	0.135	4848	10.57	2.55	1.425	0.344

These experiments are designed to show the benefits of application-specific NoC architectures. In the second set of experiments, we implemented an exact algorithm, referred to as EXACT, that exhaustively enumerates all distinct set partitions. These experiments are designed to show how close our synthesis algorithms are to exact enumeration results.

Comparison of results

The synthesis results of our four algorithms on all unicast benchmarks at 70nm with comparison to results using mesh and opt-mesh topologies are shown in Table 3.1. For each algorithm, the power results, the execution times, and power improvements of that algorithm over mesh and opt-mesh topologies are reported. The power results of all algorithms relative to mesh implementations are graphically compared in Figure 3.3(a). The results show that all algorithms can efficiently synthesize NoC architectures that minimize power consumption. All algorithms can achieve substantial reduction in power consumption over the standard mesh and opt-mesh topologies in all cases. The two heuristic algorithms can achieve comparable results as the two perturbation-based algorithms.

In particular, CLUSTER can achieve on average a 6.92$\times$ reduction in power over the standard mesh topologies and a 2.68$\times$ reduction over the optimized mesh topologies, and DECOMPOSE can achieve on average a 6.83$\times$ and a 2.60$\times$ reduction in power over the standard mesh and optimized mesh topologies. Similarly, PERTURB can achieve on average a 7.16$\times$ and a 2.73$\times$ reduction in power over the standard mesh and optimized mesh topologies, and R-PERTURB can achieve on average a 6.99$\times$ and a 2.66$\times$ reduction in power over the standard mesh and optimized mesh topologies, respectively.

The execution times of all algorithms are graphically compared in Figure 3.3(b).

Table 3.2: NoC hop count comparisons on unicast applications.

	CLUSTER		DECOMPOSE		R-PERTURB		PERTURB		mesh
Application	Avg hops	Improv. CL/mesh	Avg. hops	Improv. DE/mesh	Avg. hops	Improv. R-P/mesh	Avg. hops	Improv. PERT/mesh	Avg. hops
VOPD	0.71	3.93	0.64	4.36	0.64	4.36	0.64	4.36	2.79
MPEG4	1.38	2.17	1.46	2.05	1.38	2.17	1.38	2.17	3.00
PIP	0.63	3.57	0.63	3.57	0.63	3.57	0.63	3.57	2.25
MWD	0.54	4.70	0.54	4.70	0.54	4.70	0.54	4.70	2.54
H263	1.20	2.17	1.20	2.17	1.20	2.17	1.20	2.17	2.60
G5	0.83	3.11	0.83	3.11	0.83	3.11	0.83	3.11	2.58
mp3enc	1.00	2.43	1.00	2.43	1.00	2.43	1.00	2.43	2.43
mp3dec	0.50	4.66	0.50	4.66	0.50	4.66	0.50	4.66	2.33
h263dec	0.75	3.00	0.75	3.00	0.75	3.00	0.75	3.00	2.25
MMEnc	1.05	2.15	1.05	2.15	1.05	2.15	1.05	2.15	2.26
MMDec	1.14	2.07	1.14	2.07	1.14	2.07	1.14	2.07	2.36
MMS	1.21	1.90	0.97	2.37	0.97	2.37	1.00	2.30	2.30
V+M	1.04	2.21	1.30	1.77	1.15	2.00	1.04	2.21	2.30
M+P	0.62	3.39	0.62	3.39	0.62	3.39	0.62	3.39	2.10
V+M+M	0.93	2.40	1.00	2.23	0.98	2.28	0.90	2.48	2.23
4in1	0.90	2.46	0.94	2.35	0.92	2.40	0.92	2.40	2.21

The results show that all algorithms work quite fast. The two heuristic algorithms, CLUSTER and DECOMPOSE, work much faster than PERTURB. As can be seen, CLUSTER can achieve better results than DECOMPOSE because it examines more set partition candidates in its solution space, but it requires longer run times. R-PERTURB also works faster than PERTURB because it searches a smaller state space than PERTURB. However, it can achieve similar results as PERTURB, with execution times comparable to the two heuristic algorithms.

To evaluate the performance of the synthesized topologies, average hop count results for the benchmarks from the synthesized topology are reported in Table 3.2 and graphically compared in Figure 3.3(c). Hop counts correspond to the number of intermediate routers that a packet needs to pass through from the source to the destination. The results show that the solutions obtained using our algorithms can achieve much lower hop counts, and thus lower latencies, than the corresponding mesh topologies. In particular, CLUSTER, DECOMPOSE, R-PERTURB and PERTURB can achieve on average a 2.89$\times$, 2.90$\times$, 2.93$\times$, and 2.95$\times$ reduction in hop count. In a number of benchmarks, some modules only have single incoming flow or single outgoing flow. For example, for the VOPD benchmark, 6 out of the 12 modules have at most one incoming flow as well as one outgoing flow, and 10 out of the 12 modules have either at most one outgoing flow or one incoming

Table 3.3: NoC router area comparisons on unicast applications.

	CLUSTER		DECOMPOSE		R-PERTURB		PERTURB		opt-mesh
Application	Router Area (mm^2)	Impro. CL/ opt-mesh	Router Area (mm^2)	Impro. DE/ opt-mesh	Router Area (mm^2)	Impro. R-P/ opt-mesh	Router Area (mm^2)	Impro. PERT./ opt-mesh	Router Area (mm^2)
VOPD	0.16	6.28	0.16	6.28	0.16	6.28	0.16	6.28	1.01
MPEG4	0.21	2.28	0.21	2.28	0.21	2.28	0.21	2.28	0.48
PIP	0.10	4.00	0.10	4.00	0.10	4.00	0.10	4.00	0.41
MWD	0.16	5.29	0.14	6.35	0.14	6.35	0.11	7.95	0.87
H263	0.35	1.37	0.21	2.31	0.21	2.31	0.21	2.31	0.48
G5	0.20	3.59	0.20	3.59	0.20	3.59	0.20	3.59	0.71
mp3enc	0.16	2.32	0.16	2.32	0.16	2.32	0.16	2.32	0.36
mp3dec	0.10	2.94	0.10	2.94	0.10	2.94	0.10	2.94	0.31
h263dec	0.18	2.47	0.18	2.47	0.18	2.47	0.18	2.47	0.43
MMEnc	0.53	1.71	0.45	2.02	0.45	2.02	0.45	2.02	0.91
MMDec	0.35	2.58	0.35	2.58	0.35	2.58	0.31	2.87	0.89
MMS	0.95	1.91	0.82	2.22	0.79	2.29	0.85	2.14	1.81
V+M	0.54	2.18	0.53	2.23	0.36	3.34	0.48	2.47	1.19
M+P	0.31	3.30	0.31	3.30	0.31	3.30	0.31	3.30	1.03
V+M+M	0.75	2.41	0.75	2.41	0.75	2.41	0.75	2.41	1.81
4in1	0.86	2.36	0.86	2.36	0.86	2.36	0.79	2.54	2.01

flow. For these benchmarks, the most efficient architectures are actually ones that provide direct network links between network interfaces for some of its traffic flows without going through intermediate routers[2]. For these benchmarks, the average hop count may be less than one since not all flows necessarily pass through intermediate routers. Our flow partitioning problem formulation is able to arrive at these implementations by exploring set partitions in which some flows are grouped in their own partition.

To evaluate the area costs of the synthesized solutions, we also used Orion [31] to estimate the areas of the routers in the synthesized architectures, using the same 70nm technology used for power estimation. The area cost of a solution corresponds to the sum of the router areas in the solution. The results are presented in Table 3.3 and compared in Figure 3.3(d). Total area costs of all routers in a custom NoC solution produced by our algorithms are only in the range of 0.10 to 0.86 mm^2, even for the largest benchmark 4in1 with 44 modules (about 0.02 mm^2 amortized area costs per module in the 4in1 example,

[2]State-of-the-art router microarchitectures, such as those proposed in [35, 36, 37], employ finite buffers and virtual channels. Flow control is used to prevent upstream routers or network interfaces from sending more data when either buffer space or virtual channel is unavailable. Network interfaces that interoperate with these router microarchitectures must also correspondingly support the same flow control mechanism. This flow control mechanism can be used to control data transfers between network interfaces that are directly connected by a network link. Our synthesis algorithms can also be constrained to produce architectures where flows are required to pass through at least one router.

which is small in comparison to the expected size of modules). In comparisons to the area costs of the opt-mesh solutions, our algorithms are on average $3.12\times$ lower.

In the next set of experiments, we compare our algorithms with an exact algorithm that enumerates all distinct set partitions. As the number of distinct set partitions grows $\Theta(n \log n)^n$, the CPU times for generating the exact solutions increase very quickly. We set a CPU timeout limit of 8 hours. The results are compared in Table 3.4. Out of the 16 benchmarks tested, we were able to obtain results for exact enumeration for only 6 of the benchmarks. The largest unicast benchmark that exact enumeration could complete was a benchmark with 12 flows (namely G5). This is because the number of set partitions of N flows grows as the Bell number, and $B_{12} = 4,213,597$ and $B_{13} = 27,644,437$. The exact enumeration of G5 with 12 flows took about 4 hours. However, we estimate that it will take more than 27 hours to generate exact solution for a benchmark with 13 flows (e.g. MPEG4), which is well beyond our 8 hour timeout limit.

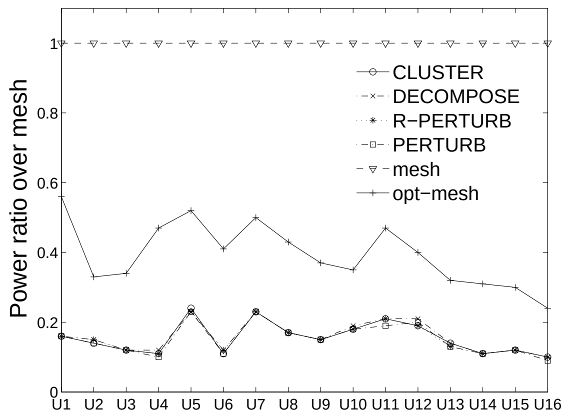
On the other hand, as shown in Table 3.4, of the 6 benchmarks where exact enumeration was possible, our PERTURB algorithm could achieve the exact solution in all cases and the longest execution time was only about 2 minutes. CLUSTER was able to achieve the same results as exact enumeration in 5 out of the 6 cases, and on average, the results are within just 1% of the exact results. Moreover, the CPU times for the 6 benchmarks were all under 1 second whereas the EXACT algorithm took as much as 4.5 hours to achieve similar results. Likewise, DECOMPOSE and R-PERTURB were able to achieve the exact solution in 4 out of the 6 cases, and on average, the results are within 2% of the exact results, and these results were achieved in the range of 0.23 to 7 seconds.

3.7.3 Results for Multicast Applications

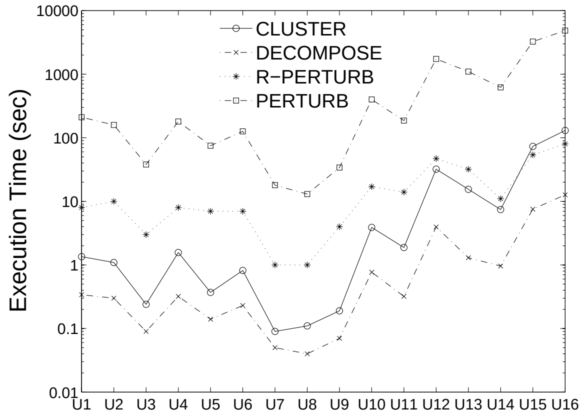
In this section, we present experimental results on benchmarks with multicast traffic to evaluate the performance of our algorithms on the synthesis of NoC communication architectures with multicast routing.

Benchmark generation using Rent's rule

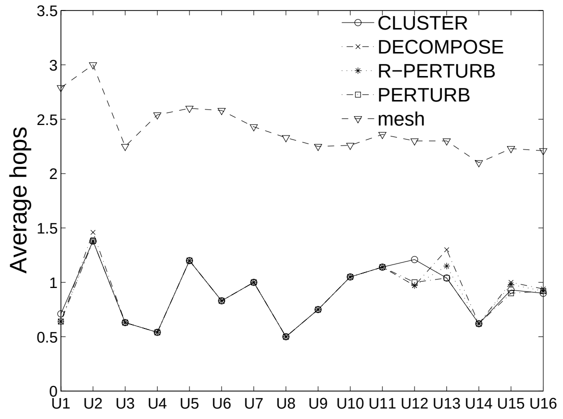
To generate synthetic benchmarks with multicast traffic, we used the NoC-centric bandwidth-version of Rent's rule proposed by Greenfield et al. [46]. In particular, they



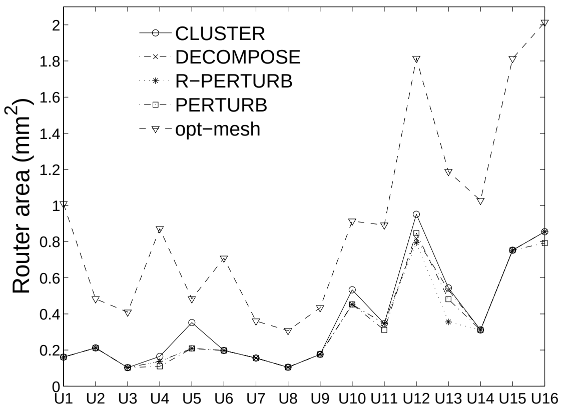
(a) Power



(b) Execution time



(c) Hop count



(d) Area

Figure 3.3: Comparisons of all algorithms on unicast applications.

Table 3.4: NoC power comparisons with exact solutions on unicast applications.

	CLUSTER			DECOMPOSE			R-PERTURB			PERTURB			EXACT	
Appli.	Power (W)	Time (sec)	Power Ratio CL/EX	Power (W)	Time (sec)	Power Ratio DE/EX	Power (W)	Time (sec)	Power Ratio R-P/EX	Power (W)	Time (sec)	Power Ratio PERT/EX	Power (W)	Time (sec)
VOPD	0.042	1.35	n.a.	0.042	0.34	n.a.	0.042	8	n.a.	0.042	210	t.o.	t.o.	t.o.
MPEG4	0.039	1.09	n.a.	0.040	0.30	n.a.	0.040	10	n.a.	0.039	159	t.o.	t.o.	t.o.
MWD	0.028	1.57	n.a.	0.031	0.32	n.a.	0.028	8	n.a.	0.026	180	t.o.	t.o.	t.o.
MMEnc	0.068	3.89	n.a.	0.073	0.77	n.a.	0.068	17	n.a.	0.068	400	t.o.	t.o.	t.o.
MMDec	0.054	1.88	n.a.	0.054	0.32	n.a.	0.054	14	n.a.	0.049	186	t.o.	t.o.	t.o.
MMS	0.123	32.07	n.a.	0.132	3.96	n.a.	0.121	47	n.a.	0.126	1735	t.o.	t.o.	t.o.
V+M	0.086	15.47	n.a.	0.090	1.30	n.a.	0.082	32	n.a.	0.081	1100	t.o.	t.o.	t.o.
M+P	0.053	7.45	n.a.	0.053	0.96	n.a.	0.053	11	n.a.	0.053	620	t.o.	t.o.	t.o.
V+M+M	0.120	73.02	n.a.	0.124	7.54	n.a.	0.122	54	n.a.	0.119	3248	t.o.	t.o.	t.o.
4in1	0.139	130.50	n.a.	0.142	12.68	n.a.	0.141	80	n.a.	0.135	4848	t.o.	t.o.	t.o.
mp3enc	0.023	0.09	**1.00**	0.023	0.05	**1.00**	0.023	1	**1.00**	0.023	18	**1.00**	0.023	1
mp3dec	0.016	0.11	**1.00**	0.016	0.04	**1.00**	0.016	1	**1.00**	0.016	13	**1.00**	0.016	1
h263dec	0.026	0.19	**1.00**	0.026	0.07	**1.00**	0.026	4	**1.00**	0.026	34	**1.00**	0.026	7
PIP	0.021	0.24	**1.00**	0.021	0.09	**1.00**	0.021	3	**1.00**	0.021	38	**1.00**	0.021	9
H263	0.037	0.37	**1.05**	0.036	0.14	**1.02**	0.036	7	**1.02**	0.035	75	**1.00**	0.035	293
G5	0.028	0.82	**1.00**	0.031	0.23	**1.10**	0.031	7	**1.09**	0.028	126	**1.00**	0.028	14440

showed that the traffic distribution models of NoC applications should follow a similar Rent's rule distribution as in conventional VLSI netlists. The bandwidth-version of Rent's rule was derived showing that the relationship between the external bandwidth B across a boundary and the number of blocks G within a boundary obeys $B = kG^{\beta}$, where k is the average bandwidth for each block and β is the Rent's exponent. The benchmark generation procedure proposed in [47] is adopted and modified in accordance to NoC-centric Rent's rule to generate multicast benchmarks. The average bandwidth k for each block and Rent's exponent β are specified by the user. In our experiments, we generated large NoC benchmarks by varying k ranging from 100kb/s to 500kb/s and varying β from 0.65 to 0.75. We formed multicast traffic with varying group sizes for about 10% of the flows. Thus our multicast benchmarks cover a large range of applications with mixed unicast/multicast flows and varying hop count and data rate distributions.

Comparison of results

Using the above benchmark generation process, we generated 8 multicast benchmarks with the number of modules ranging from 4 to 36 and the number of flows ranging from 6 to 84 (with some as multicast flows). We applied our four algorithms to derive optimized NoC architectures for them with the goal of minimizing power consumption. We again compare our results with both mesh and opt-mesh implementations. For multicast

Table 3.5: NoC power comparisons on multicast applications.

			CLUSTER				DECOMPOSE				R-PERTURB				PERTURB				mesh	opt-mesh
Appli.	|V|	|E|	Power (W)	Time (sec)	Improv. CL/ mesh	Improv. CL/opt	Power (W)	Time (sec)	Improv. DE/mesh	Improv. DE/opt	Power (W)	Time (sec)	Improv. R-P/ Mesh	Improv. R-P/ opt	Power (W)	Time (sec)	Improv. PERT./ mesh	Improv. PERT./ opt	Power (W)	Power (W)
M1	4	6	0.025	0.1	2.22	1.71	0.025	0.1	2.22	1.71	0.025	4	2.22	1.71	0.025	37	2.22	1.71	0.056	0.043
M2	5	8	0.032	0.2	3.21	1.74	0.034	0.1	2.98	1.62	0.032	14	3.21	1.74	0.032	114	3.21	1.74	0.102	0.055
M3	6	10	0.045	0.4	2.31	1.80	0.045	0.1	2.31	1.80	0.044	15	2.36	1.84	0.042	203	2.49	1.94	0.104	0.081
M4	16	32	0.191	31	2.10	1.33	0.176	3	2.29	1.45	0.178	178	2.26	1.44	0.180	2840	2.24	1.42	0.402	0.255
M5	20	48	0.266	132	2.02	1.56	0.291	7	1.85	1.43	0.271	254	1.99	1.54	0.260	4208	2.07	1.60	0.538	0.417
M6	25	58	0.347	235	2.04	1.64	0.422	13	1.67	1.35	0.360	374	1.96	1.58	0.347	14137	2.04	1.64	0.707	0.570
M7	30	68	0.426	499	2.05	1.50	0.419	20	2.09	1.53	0.418	476	2.09	1.53	0.426	20980	2.05	1.50	0.875	0.639
M8	36	84	0.528	1430	2.07	1.55	0.605	39	1.81	1.35	0.536	712	2.04	1.53	0.528	28730	2.07	1.55	1.092	0.818

routing over a mesh implementation, we applied the efficient multicast routing algorithm described in [69] to determine the routing of multicast traffic. We then again generated opt-mesh implementations by eliminating router ports and links that are not used by the traffic flows.

The power consumption results and the execution times of all algorithms on the multicast benchmarks are reported in Table 3.5. The power results of all algorithms relative to mesh implementations are compared in Figure 3.4(a). The execution times of our four algorithms are compared in Figure 3.4(b). For the multicast benchmarks, all of our algorithms can achieve substantial reduction in power over the standard mesh and opt-mesh implementations. Among all algorithms, PERTURB achieves the best performance over our other algorithms, with on average a 2.30× reduction in power over the standard mesh and a 1.64× reduction over the optimized mesh topologies. R-PERTURB is slightly worse than PERTURB by achieving on average a 2.27× and a 1.61× reduction in power over the standard mesh and optimized mesh topologies. However, its execution times are 8 to 45 times faster than PERTURB. The two heuristic algorithms can achieve comparable results as the two probabilistic algorithms. In particular, CLUSTER can achieve on average a 2.25× and a 1.60× reduction in power over the standard mesh and optimized mesh topologies, and DECOMPOSE can achieve on average a 2.15× and a 1.53× reduction in power over the standard mesh and optimized mesh topologies, respectively. Both heuristic algorithms work much faster than PERTURB. For example, DECOMPOSE can obtain all results in less than 1 minute. For applications with large problem sizes, the heuristic algorithms and R-PERTURB can be used without sacrificing much performance.

Average hop count results are also reported in Table 3.6 and Figure 3.4(c). The results show that, for multicast applications, the synthesized topologies using our algorithms can also achieve lower hop counts than both mesh and opt-mesh topologies. In particular, on average, CLUSTER, DECOMPOSE, R-PERTURB and PERTURB can achieve 1.84×,

Table 3.6: NoC hop count comparisons on multicast applications.

	CLUSTER		DECOMPOSE		R-PERTURB		PERTURB		mesh
Appli.	Avg. hops	Improv. CL/mesh	Avg. hops	Improv. DE/mesh	Avg. hops	Improv. R-P/mesh	Avg. hops	Improv. PERT/mesh	Avg. hops
M1	0.90	2.22	0.90	2.22	0.90	2.22	0.90	2.22	2.00
M2	0.90	2.50	0.79	2.86	0.90	2.50	0.90	2.50	2.25
M3	1.08	2.31	1.08	2.31	1.00	2.50	1.35	1.85	2.50
M4	1.52	1.54	1.80	1.30	1.63	1.44	1.55	1.51	2.34
M5	1.94	1.53	2.01	1.47	2.25	1.32	1.75	1.70	2.96
M6	2.34	1.35	2.30	1.38	1.44	2.19	1.86	1.70	3.16
M7	1.46	2.00	2.21	1.31	1.79	1.62	2.03	1.43	2.91
M8	2.18	1.26	2.03	1.35	2.28	1.20	2.08	1.32	2.74

Table 3.7: NoC router area comparisons on multicast applications.

	CLUSTER		DECOMPOSE		R-PERTURB		PERTURB		opt-mesh
Appli.	Router Area (mm^2)	Impro. CL/ opt-mesh	Router Area (mm^2)	Impro. DE/ opt-mesh	Router Area (mm^2)	Impro. R-P/ opt-mesh	Router Area (mm^2)	Impro. PERT/ opt-mesh	Router Area (mm^2)
M1	0.18	1.77	0.18	1.77	0.18	1.77	0.18	1.77	0.31
M2	0.21	1.89	0.23	1.74	0.21	1.89	0.21	1.89	0.40
M3	0.35	1.65	0.37	1.57	0.29	2.01	0.25	2.32	0.58
M4	1.14	1.53	1.20	1.45	1.12	1.56	1.15	1.52	1.74
M5	1.59	1.78	1.61	1.76	1.81	1.56	1.56	1.81	2.83
M6	2.03	1.90	2.03	1.90	2.04	1.89	2.04	1.89	3.86
M7	2.68	1.60	2.41	1.78	2.54	1.69	2.51	1.71	4.29
M8	3.06	1.77	3.01	1.81	3.08	1.76	3.02	1.79	5.43

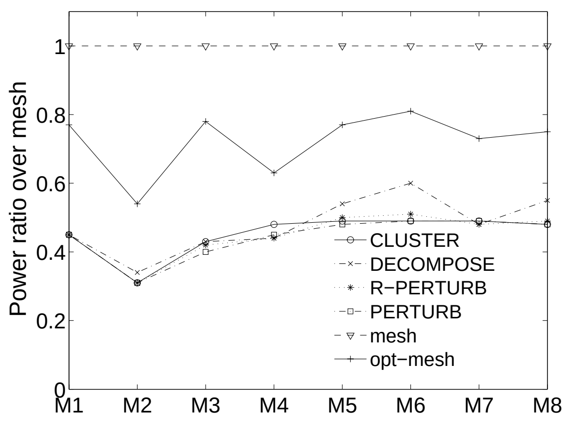
1.78×, 1.88×, and 1.78× reduction in hop count.

The area costs in terms of router areas for the multicast benchmarks are reported in Table 3.7 and Figure 3.4(d). The area costs of our algorithms are in the range from 0.18 mm^2 to 3.06 mm^2 for the largest benchmark M8 with 84 modules (about 0.04 mm^2 amortized area costs per module). In comparisons to the area costs of the opt-mesh solutions, our algorithms are on average 1.77× lower.

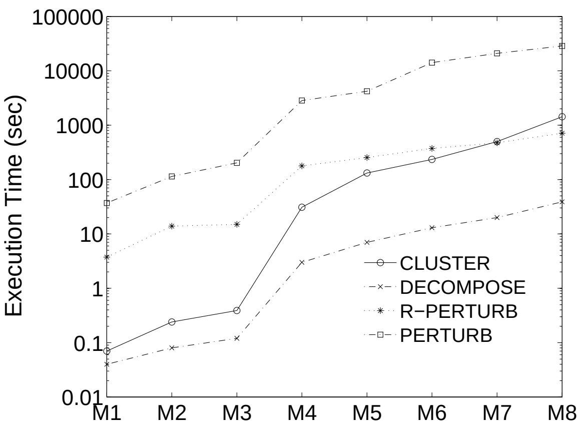
Finally, we again compare our algorithms with the exact enumeration algorithm for multicast applications. The results are compared in Table 3.8. We again set a CPU timeout limit of 8 hours. Out of the 8 multicast benchmarks tested, we were only able to obtain the results for exact enumeration for 3 of them. Of these 3 benchmarks, PERTURB could achieve the exact solution in all these cases, and the longest execution time was only about 3.5 minutes. CLUSTER and R-PERTURB were able to achieve the exact solution in 2 out of the 3 cases, and on average, the results are within just 3% and 2% of the exact results respectively. Likewise, DECOMPOSE was able to achieve the exact solution in 1 out of 3

Table 3.8: NoC power comparisons with exact solutions on multicast applications.

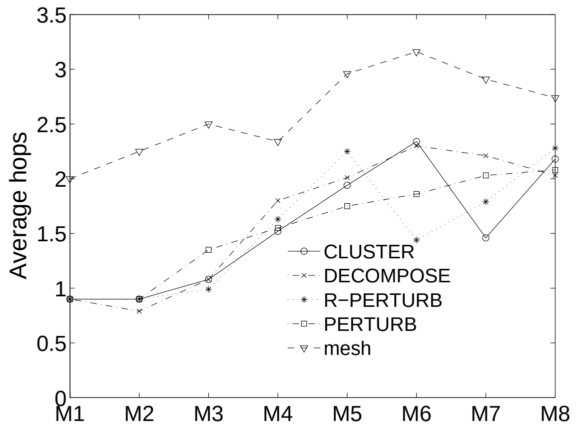
	CLUSTER			DECOMPOSE			R-PERTURB			PERTURB			EXACT	
Appli.	Power (W)	Time (sec)	Power Ratio CL/EX	Power (W)	Time (sec)	Power Ratio DE/EX	Power (W)	Time (sec)	Power Ratio R-P/EX	Power (W)	Time (sec)	Power Ratio PERT/EX	Power (W)	Time (sec)
M1	0.025	0.1	**1.00**	0.025	0.1	**1.00**	0.025	4	**1.00**	0.025	37	**1.00**	0.025	0.2
M2	0.032	0.2	**1.00**	0.034	0.1	**1.08**	0.032	14	**1.00**	0.032	114	**1.00**	0.032	7
M3	0.045	0.4	**1.08**	0.045	0.1	**1.08**	0.044	15	**1.06**	0.042	203	**1.00**	0.042	258
M4	0.191	31	n.a.	0.176	3	n.a.	0.178	178	n.a.	0.180	2840	n.a.	t.o.	t.o.
M5	0.266	132	n.a.	0.291	7	n.a.	0.271	254	n.a.	0.260	4208	n.a.	t.o.	t.o.
M6	0.347	235	n.a.	0.422	13	n.a.	0.360	374	n.a.	0.347	14137	n.a.	t.o.	t.o.
M7	0.426	499	n.a.	0.419	20	n.a.	0.418	476	n.a.	0.426	20980	n.a.	t.o.	t.o.
M8	0.528	1430	n.a.	0.605	39	n.a.	0.536	712	n.a.	0.528	28730	n.a.	t.o.	t.o.



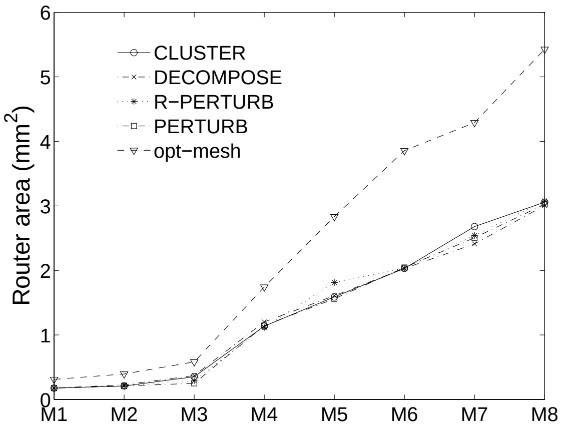
(a) Power



(b) Execution time



(c) Hop count



(d) Area

Figure 3.4: Comparisons of all algorithms on multicast applications.

cases, and on average, the results are within 5% of the exact results.

Acknowledgement

This chapter is in part a reprint of the material in the following papers: Shan Yan, Bill Lin, “Custom Networks-on-Chip Architectures with Multicast Routing”, *IEEE Transactions on VLSI Systems*, volume: 17, issue: 3, on pages: 342-355, March 2009; Shan Yan, Bill Lin, “Application-specific Network-on-Chip architecture synthesis based on set partitions and Steiner Trees”, *13th Asia and south Pacific Design automation conference (ASP-DAC 2008)*, 2008: 277-282. The dissertation author was the primary author of both papers.

Chapter 4

Design Custom Topologies Based on Rip-Up and Reroute

4.1 Overview

In this chapter, we describe an alternative NoC synthesis approach based on a *rip-up and reroute* concept that has been successfully used in the VLSI routing problem [48, 49, 50]. The rip-up and reroute concept provides us with a heuristic iterative mechanism to identify increasingly improving solutions. There are two central differences between our on-chip network routing and design problem and the VLSI routing problem. The first is the ability to share network resources in our problem, and the second is the difference in cost models. In the latter case, the costs of routers and links are not simple linear costs, and the sharing of network resources further complicates the optimization process.

The algorithms presented in this chapter improve over the earlier algorithms presented in Chapter 3. Our earlier NoC synthesis algorithms were based on the formulation of the problem as set partitioning of traffic flows, finding a good network topology for each flow set using a Steiner tree formulation and providing an optimized network implementation for the derived topologies. All possible set partition of flows are investigated in an intelligent way and a Rectilinear Sterner tree problem is solved for each intermediate set partition, which makes those algorithms less efficient for future large applications with hundreds to thousands of cores envisioned.

In particular, we propose a very efficient algorithm called Ripup-Reroute-and-

Router-Merging (RRRM) that synthesizes custom NoC architectures for supporting both unicast and multicast traffic flows. The key part of the algorithm is a rip-up and reroute procedure that routes multicast flows by way of finding the optimum multicast tree on a condensed multicast routing graph using the directed minimum spanning tree formulation and the efficient algorithms [51, 52]. Then a router merging procedure follows after to further optimize the network topology

As already has been shown in the previous chapter, our Steiner-tree based formulation already significantly outperformed regular mesh and optimized mesh topologies. In comparison to those algorithms, the performance of our new algorithm was able to achieve a relative reduction of up to 45% in terms of power consumption, up to 21% in terms of hop counts and up to 39% in terms of router area. More important, the execution times of our new algorithm are 2 to 3 orders of magnitude faster than the previous algorithms even for very large benchmarks.

The rest of this chapter is organized as follows. Section 4.2 describes the details of RRRM algorithm and Section 4.3 presents the experimental results.

4.2 Design Algorithms – RRRM

In this section, we present algorithms for the NoC topology synthesis process based on the rip-up and reroute approach. The entire process is a joint multicast routing and network design procedure that consists of the inter-related steps of constructing an initial network topology, rip-up and rerouting multicast flows to design the network topology, inserting the corresponding network links and router ports to implement the routing, and merging routers to optimize network topology based on design objectives. In particular, we propose an algorithm called Ripup-Reroute-and-Router-Merging (RRRM). The details of the algorithm are discussed in this section.

4.2.1 Initial network construction

The details of RRRM are described in Algorithm 6. RRRM takes a communication demand graph (CDG) and an evaluation function as inputs and generates an optimized network architecture as output. It starts with initializing a network topology by a simple router allocation and flow routing scheme. Then it uses a procedure of rip-up and rerouting flows to refine and optimize the network topology. After that, a router merging step is done to further optimize the topology to obtain the best result.

In the initialization, every flow is routed using its own network. To construct an initial network topology, a router is allocated at each core and placed close to the location of network interface. These routers are not actual routers that will be included in the network topology. Only those that have traffic either multiplexed from more than two ports to the same port or de-multiplexed from one port to more than two ports at the end of the RRRM procedure will be included. After router allocation, a Routing Cost Graph (RCG) is generated (Algorithm 6 line 2). RCG is a very important graph used in the whole rip-up and reroute procedure of the RRRM algorithm.

Definition 2. *The $RCG(R, E)$ is a weighted directed complete graph with each vertex $r_i \in R$ represents a router, and each directed edge $e_{ij} = (r_i, r_j) \in E$ from r_i to r_j corresponds to a connection from r_i to r_j. A weight $w(e_{ij})$ is attached to each edge which represents the incremental cost of routing a flow f through e_{ij}.*

Please note that RCG does not represent the actual physical connectivity between different routers and its edge weights change during the whole RIPUP-REROUTE procedure for different flows. Also, the actual physical connectivity between the routers is established during RIPUP-REROUTE procedure, which is explained in the following sections.

Before RIPUP-REROUTE, initial network topology is constructed using InitialNetworkConstruction() procedure. Each flow $e_k = (s_k, d_k)$ in the CDG is routed using a direct connection from router r_{s_k} to router r_{d_k}, where r_i is the router that core i connects to, and the path is saved in $path(e_k)$. Multicast flows are routed as a sequence of unicast flows from the source to each of their destinations. The links and router ports are configured and saved. If a connection between routers can not meet the delay constraints, its corresponding edge weight in RCG is set to infinity. This can be used to guide the rerouting of the flows to use other valid links instead of this one in the RIPUP-REROUTE procedure.

As an example, after initial network construction, the connectivity of routers for the example shown in Figure 2.3(a) is shown in Figure 4.1(a).

Algorithm 6 RRRM($G(V, E, \pi, \lambda), C, L$)

```
Input: $G(V, E, \pi, \lambda)$: CDG, $C$: cost function,
    $L$: library of network components
Output: $T$: synthesized network topology
1: $R$ = InitialRouterAllocation($G$)
2: $RCG$ = ConstructFullyConnectedGraph($R$)
3: ($links, routers$) = InitialNetworkConstruction($G, RCG$)
4: ($links, routers$) = RIPUP-REROUTE($G, RCG, C, L$)
5: $cost$ = EvaluatePowerConsumption($links, routers$)
6: ROUTER-MERGING($cost, links, routers$)
7: $T$ = ObtainBestTopology($links, routers$)
8: return $T$

InitialNetworkConstruction($G, RCG$)
1: for all flow $e_k = (s_k \rightarrow D_k) \in E$ do
2:   for all destination $d_{ki} \in D_k$ do
3:     route $e_k$ using a direct connection between $r_{s_k}$ and $r_{d_{ki}}$
4:     if $link(r_{s_k}, r_{d_{ki}})$ exists, update bandwidths of links, routers ports
5:     $path(e_k) = (r_{s_k} \rightarrow r_{d_{ki}})$
6:     update routers $r_{s_k}, r_{d_{ki}}, link(r_{s_k}, r_{d_{ki}})$
7:     if $link(r_{s_k}, r_{d_{ki}})$ not meet delay constraint, update $wgt(r_{s_k}, r_{d_{ki}}) = \infty$ in $RCG$
8:   end for
9: end for
```

4.2.2 Flow Ripup and Rerouting

Once the initial network is constructed and the initial flow routing is done, the key procedure of the algorithm – RIPUP-REROUTE procedure is invoked to route flows and find an optimized network topology.

The details of RIPUP-REROUTE are described in Algorithm 7. In the RIPUP-REROUTE procedure, each multicast routing step is formulated as a minimum directed spanning tree problem. Two important graphs, Multicast Routing Graph (MRG) and Multicast Routing Tree (MRTree), are used to help facilitate the rip-up and rerouting procedure. They are defined as follows.

Definition 3. *Let f be a multicast flow with source $s \in V$ and one or more destinations $D \subseteq V$. i.e., $D = \{d_1, d_2, \ldots, d_{|D|}\}$, each $d_i \in V$.*

A Multicast Routing Graph (MRG) is a complete graph $\Gamma(N, A)$ defined for f as follows:

- *$N = s \cup D$.*
- *There is a directed arc between every pair of nodes (i, j) in N. Each arc $a_{i,j} \in A$ corresponds to a shortest path $p_{i,j}$ between the same nodes in the corresponding RCG, $p_{i,j} = e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_k$.*
- *The weight for arc $a_{i,j}$, $w(a_{i,j})$, corresponds to the* path weight *of the corresponding shortest path $p_{i,j}$ in RCG. i.e.,*

$$w(a_{i,j}) = \sum_{e_i \in p} w(e_i)$$

Definition 4. *A Multicast Routing Tree (MRTree) is the* Minimum Directed Spanning Tree *for multicast routing graph $\Gamma(N, A)$ with $s \in N$ as the root.*

When a flow is ripped-up and rerouted, its current path is deleted and the links and router ports resources it occupies are released (line 3). Then based on the current network connectivity and resources occupation, the RCG related to this flow is built and the weights of all edges in RCG are updated (line 4). In particular, for every pair of routers in RCG, the cost of using those routers and the link connecting them is evaluated. This cost depends on the sizes of the routers, the traffic already routed on the routers and the connectivity of the routers to other routers. It also depends on whether an existing physical link will be used or a new physical link needs to be installed. If there are already router ports and links that can support the traffic, the marginal cost of reusing those resources is calculated. Otherwise, the cost of opening new router ports and installing new physical link to support the traffic is calculated. The cost is assigned as edge weight to the edge connecting the pair of routers in RCG. If the physical links used to connect the routers can not satisfy the delay constraints, a weight of infinity is assigned to the corresponding edges in RCG.

Once the RCG is constructed, the multicast routing graph (MRG) for the flow is generated from RCG (line 5). MRG is built by including every source and destination router of the flow as its nodes. For each pair of the nodes in MRG, the least cost directed path with least power consumption on RCG is found for the corresponding routers using Dijkstra's shortest path algorithm and the cost is assigned as edge weight to the edge connecting the two nodes in MRG. Then the Chu-Liu/Edmonds algorithm [51, 52] is used to find the rooted directed minimum spanning tree of MRG with the source router as root. A rooted directed spanning tree of a graph is defined as a graph which connects, without any cycle, all n nodes in the graph with $n - 1$ arcs such that the sum of the weight of all the arcs is minimized. Each node, except the root, has one and only one incoming arc. This directed minimum spanning tree is obtained as the multicast routing tree (MRTree) so that the routes of the multicast flow follows the structure of this tree. The details of Chu-Liu/Edmonds Algorithm is summarized in Algorithm 8. The multicast routing for flow f in RCG can be obtained by projecting MRTree back to RCG by expanding the corresponding arcs to paths. A special case is when f is a *unicast* flow with source s and destination d. In this case, MRG will just consist of two nodes, namely s and d, and one directed arc from s to d. Therefore, the routing between s and d in RCG is simply a shortest path between s and d.

After the path is determined, the routers and links on the chosen path are updated.

As an example, Figure 4.1(b) shows the RCG for rerouting the multicast flow e_7. For clarity, only part of the edges are shown for RCG. The MRG and MRTree for e_7 are shown in Figure 4.1(c) and (d) respectively. By projecting MRTree back to RCG, the routing path for e_7 is determined, namely e_7 bifurcates in the source router R_4 to reach R_6 and v_6, then it is transferred over the network link between R_4 to R_2 to reach v_2, and then bifurcates to reach R_5 and v_5. The real physical connectivity between routers before and after rip-up and rerouting e_7 are also shown in Figure 4.1(e) and (f). From them, we observe that the link between R_4 and R_5 and their corresponding ports are saved thus the power consumptions are reduced after rerouting e_7 by utilizing the existing network resources for routing other flows.

This RIPUP-REROUTE process is repeated for all the flows. The results of this procedure depends on the order that the flows are considered, so the entire procedure can be repeated for several times to reduce the dependency of the results on flow ordering [1]. Once the path of each flow is decided, the size of each router, the links that connect the routers are determined. Routers that have no traffic multiplexing or de-multiplexing are deleted and links are reconnected. The remain routers and links constitute the network topology. The total implementation cost of all the routers and links in this topology is evaluated and the network topology is obtained.

4.2.3 Router Merging

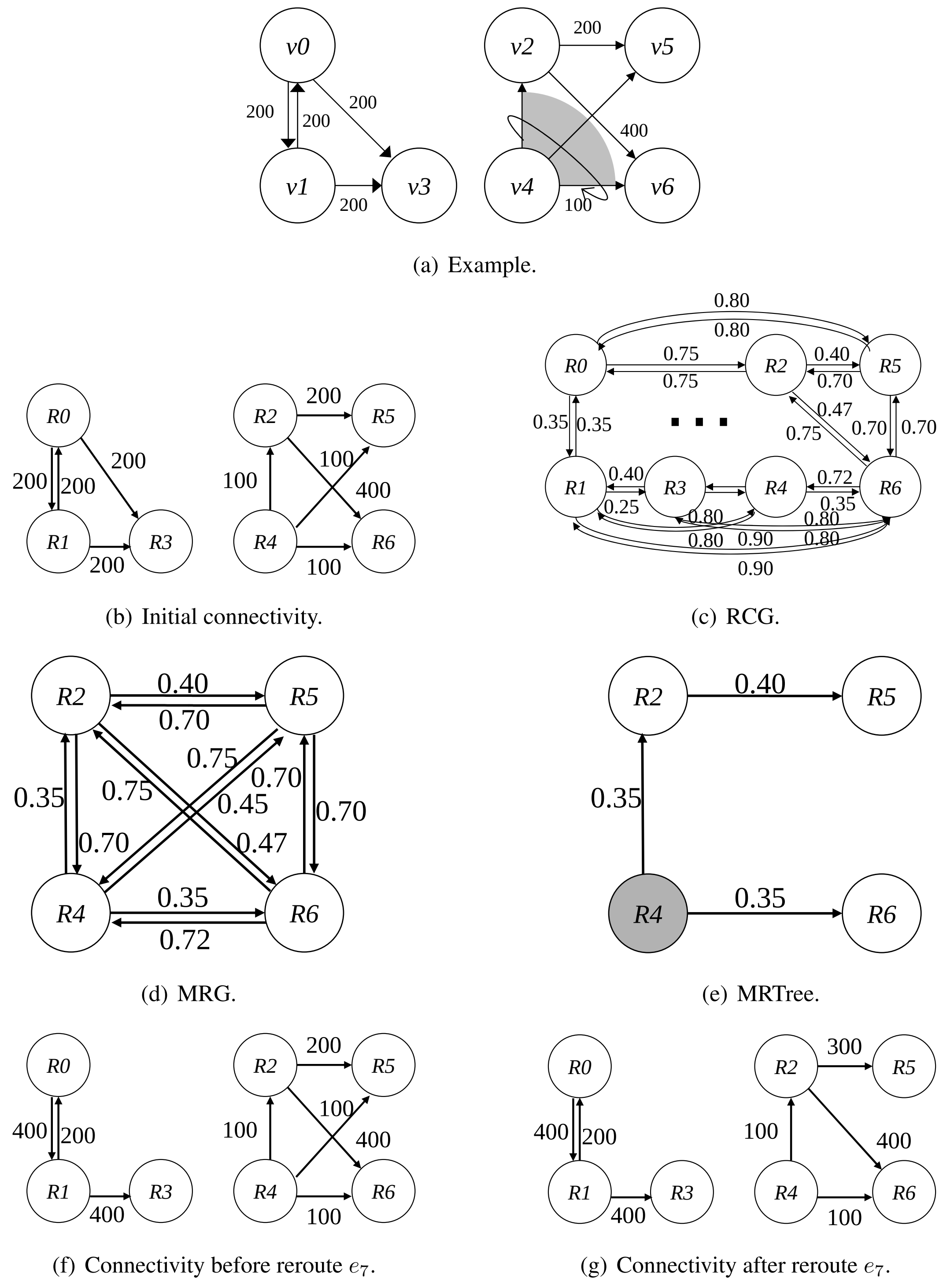
After the physical network topology has been generated using RIPUP-REROUTE, a router merging step is used to further optimize the topology to reduce the power consumption cost. The same router merging algorithm as described in Section 3.6 and Algorithm 5 is used in RRRM.

As an example, the connectivity graphs before and after ROUTER-MERGING procedure for the example of Figure 4.1(a) are shown in Figure 4.2(a) and (b). It is shown that after router merging, the network resources are reduced from 4 routers to 3 routers and the total power consumption is reduced as well.

4.2.4 Complexity of the algorithm

For an application with $|V|$ IP cores and $|E|$ flows, the initial network construction step needs $O(|E|)$ time. In the rip-up and reroute procedure, each flow is ripped-up and rerouted once. The edge weight calculation for router cost graph takes $O(|V|^2)$.

[1]In the experiments, we've tried several flow ordering strategies such as largest flow first, smallest flow first, random ordering etc., and we found the ordering of smallest flow first gave the best results. Thus we used this ordering in our experiments. Also, we observed that repeating the whole RIPUP-REROUTE procedure twice is enough to generate good results.



(a) Example.

(b) Initial connectivity.

(c) RCG.

(d) MRG.

(e) MRTree.

(f) Connectivity before reroute e_7.

(g) Connectivity after reroute e_7.

Figure 4.1: Illustration of the RIPUP-REROUTE procedure.

Algorithm 7 RIPUP-REROUTE($G(V, E, \pi, \lambda), RCG, C, L$)

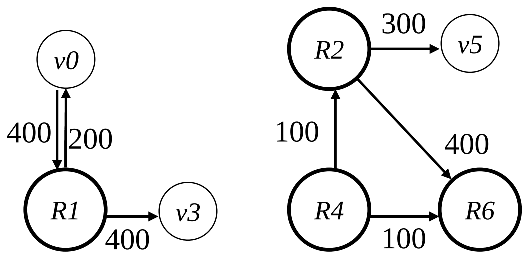
```
Input: $G(V, E, \pi, \lambda)$: CDG, $RCG$: router cost graph,
    $C$: cost function, $L$: library of network components
Output: $links, routers$: links and routers of synthesized network topology
 1: while need another round do
 2:   for all flow $e_k \in E$ in increasing order of $\lambda(e_k)$ do
 3:     delete $path(e_k)$ and release the link and router resources it occupied
 4:     update all edge weights in RCG for flow $e_k$, according to power consumption of
        the corresponding links and routers resources
 5:     $MRG(e_k)$ = ConstructMulticastRoutingGraph($RCG$)
 6:     $MTree(e_k)$ = FindMulticastTree($MRG(e_k)$)
 7:     $path(e_k)$ = Find paths from $s_k$ to $d_{ki} \in D_k$ in $MTree(e_k)$
 8:     Update $link, BW\_avail, routers$ for $path(e_k)$
 9:   end for
10: end while
11: return $links, routers$
```

Algorithm 8 DirectedMinimumSpanningTree($G(N, A)$)

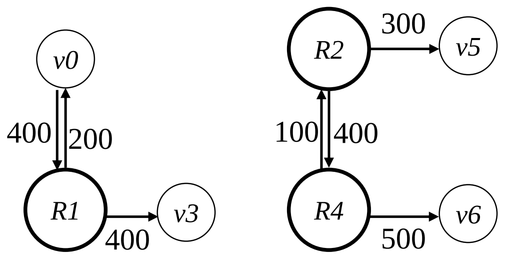
```
1: Discard the arcs entering the root if any; For each node other than the root, select the
   entering arc with the smallest cost; Let the selected $n - 1$ arcs be the set $S$.
2: If no cycle formed, $G(N, S)$ is a MST. Otherwise, continue.
3: For each cycle formed, contract the nodes in the cycle into a pseudo-node ($k$), and
   modify the cost of each arc which enters a node ($j$) in the cycle from some node ($i$)
   outside the cycle according to the following equation.

       $c(i, k) = c(i, j) - (c(x(j), j) - min_j(c(x(j), j))$

   where $c(x(j), j)$ is the cost of the arc in the cycle which enters $j$.
4: For each pseudo-node, select the entering arc which has the smallest modified cost;
   Replace the arc which enters the same real node in $S$ by the new selected arc.
5: Go to step 2 with the contracted graph.
```



(a) Before router merging.



(b) After router merging.

Figure 4.2: Illustration of the ROUTER-MERGING procedure.

For a multicast flow with m destinations, the construction of multicast routing graph takes $O((m+1)^2|V|^2)$ by finding shortest path between each pair of nodes. Then it takes $O(|V|^2)$ to find the rooted directed minimum spanning tree as the multicast tree by using the Chu-Liu/Edmonds algorithm. So the overall complexity of our algorithm is $O(|E||V|^2)$.

4.3 Experimental Results

4.3.1 Experimental Setup

We have implemented our proposed algorithm RRRM in C++. We also use Parquet [44] for the initial floorplanning step.

In all our experiments, we aim to evaluate the performance of our algorithm RRRM on all benchmarks with the objective of minimizing the total power consumption of the synthesized NoC architectures. We use the same power-performance simulator Orion and applied the same design parameters as described in Chapter 3.

As already has been shown in the previous chapter, our previous Steiner-tree based formulation and the proposed four algorithms already significantly outperformed regular mesh and optimized mesh topologies. Specifically, the two heuristic algorithms CLUSTER and DECOMPOSE could achieve similar results as the other probabilistic algorithms but with faster execution times.

Therefore, in the experiments in this chapter, in order to evaluate the effectiveness of our new algorithm, we applied RRRM on the same sets of benchmarks used in Chapter 3 and compared its synthesis results with the results of CLUSTER and DECOMPOSE. We do not repeat here the comparisons with mesh-based topologies since our new formulation already outperforms our earlier work. In particular, in order to emphasize the benefit and efficiency of our new algorithm on large benchmarks, we pick up those benchmarks with the number of cores larger than 15 and reported their results in this chapter. The results show that the algorithm RRRM outperforms CLUSTER and DECOMPOSE in both power consumption and performance with execution times two to three orders of magnitude faster. The details of the results are discussed in the following sections.

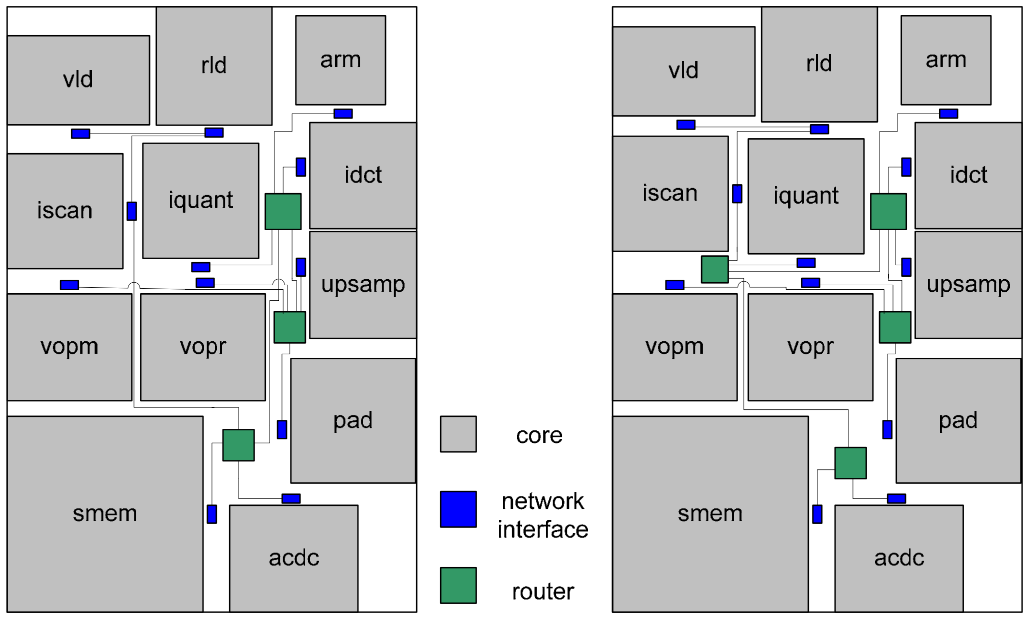
The same two groups of benchmarks were used. The first group of benchmarks was used to evaluate the performance of our algorithm on applications with only unicast flows and the second group of benchmarks was used for applications with both unicast and multicast flows.

All experimental results were obtained on a 1.5 GHz Intel P4 processor machine with 512 MB of memory running Linux.

4.3.2 Comparison of results

The floorplans for the custom topologies synthesized by our tool using different algorithms for one of the benchmark VOPD are shown in Figure 4.3. Figure 4.3(a) shows the topology generated by RRRM. It consists of three routers with $0.040W$ power consumption. Figure 4.3(b) shows the topology generated by CLUSTER and DECOMPOSE. Those two algorithms generated the same topology for VOPD consisting of four routers with each having a smaller size. Its total power consumption is $0.042W$. Although the topology generated by RRRM has larger routers, it benefits from reducing one router, leading to lower power for the overall network.

The synthesis results of our algorithm RRRM on all benchmarks at 70nm with comparison to results using CLUSTER and DECOMPOSE are shown in Table 4.1. For all benchmarks, the power results and the execution times of each algorithm, and power ratios and execution time ratios of CLUSTER and DECOMPOSE over RRRM are reported. The power results of all algorithms relative to RRRM are graphically compared in Figure 4.4(a). The results show that RRRM can efficiently synthesize NoC architectures that minimize power consumption as well as achieve good performance. Among all 14 benchmarks tested, RRRM can achieve better results than CLUSTER and DECOMPOSE for 12



(a) Topology by RRRM (b) Topology by CLUSTER and DECOMPOSE

Figure 4.3: VOPD custom topology floorplans synthesized by different algorithms

benchmarks. On average RRRM can achieve a 9% reduction in power consumption over CLUSTER and a 17% reduction in power consumption over DECOMPOSE, respectively.

Moreover, due to the low complexity of RRRM, it works much faster and more efficient than CLUSTER and DECOMPOSE. The execution times of all algorithm relative to RRRM are graphically compared in Figure 4.4(b). As can be seen from the results, RRRM can obtain results for all benchmarks under 1 minute. Even for the largest benchmarks tested with 64 cores and 164 flows, RRRM can finish within 35 seconds while it takes CLUSTER over 5 hours to finish. On average RRRM is 1786 times faster than CLUSTER and 57 times faster than DECOMPOSE. Its low complexity and very short execution time makes RRRM more suitable and efficient for benchmarks with large sizes.

To evaluate the performance of the synthesized topologies, average hop count results for the benchmarks from the synthesized topology are reported in Table 4.2 and the results of all algorithms relative to RRRM are graphically compared in Figure 4.4(c). Hop counts correspond to the number of intermediate routers that a packet needs to pass through from the source to the destination. The results show that RRRM can improve performance of the synthesized topologies as well. In particular, the solutions obtained using RRRM

Table 4.1: NoC power and execution time results

				RRRM		CLUSTER				DECOMPOSE			
Appli.	Label	|V|	|E|	Power (W)	Time (sec)	Power (W)	Ratio / RRRM	Time (sec)	ratio / RRRM	Power (W)	Ratio / RRRM	Time (sec)	Ratio / RRRM
MMS	B1	25	33	0.138	0.01	0.123	0.89	32	3207	0.132	0.96	0.97	97
V+M	B2	24	27	0.091	0.01	0.086	0.94	15	1547	0.090	0.99	1.30	130
M+P	B3	20	21	0.043	0.01	0.053	1.25	7	745	0.053	1.25	0.62	62
V+M+M	B4	36	40	0.120	0.02	0.120	1.00	73	3651	0.124	1.03	1.00	50
4in1	B5	44	48	0.134	0.02	0.139	1.04	131	6525	0.142	1.06	0.94	47
M1	B6	16	32	0.150	0.03	0.191	1.27	31	1033	0.176	1.17	3	100
M2	B7	20	48	0.257	0.06	0.266	1.03	132	2200	0.291	1.13	7	117
M3	B8	25	58	0.305	0.18	0.347	1.14	235	1306	0.422	1.38	13	72
M4	B9	30	68	0.352	0.63	0.426	1.21	499	792	0.419	1.19	20	32
M5	B10	36	84	0.470	1.52	0.528	1.12	1430	941	0.605	1.29	39	26
M6	B11	42	100	0.534	3.07	0.597	1.12	3123	1017	0.774	1.45	68	22
M7	B12	49	122	0.732	5.82	0.763	1.04	5385	925	0.868	1.19	126	22
M8	B13	56	136	0.876	15.00	0.930	1.06	8808	587	0.960	1.10	202	13
M9	B14	64	164	0.924	35.00	1.052	1.14	18576	531	1.131	1.22	344	10

Table 4.2: NoC hop counts results.

		RRRM	CLUSTER		DECOMPOSE	
application	label	Avg. Hops	Avg. Hops	Ratio /RRRM	Avg. Hops	Ratio /RRRM
MMS	B1	1.15	1.21	1.05	0.97	0.84
V+M	B2	1.07	1.04	0.97	1.30	1.21
M+P	B3	0.52	0.62	1.19	0.62	1.19
V+M+M	B4	0.90	0.93	1.03	1.00	1.11
4in1	B5	0.83	0.90	1.08	0.94	1.13
M1	B6	1.59	1.52	0.96	1.80	1.13
M2	B7	1.94	1.94	1.00	2.01	1.04
M3	B8	2.36	2.34	0.99	2.30	0.97
M4	B9	1.47	1.46	0.99	1.90	1.29
M5	B10	2.63	2.18	0.83	2.03	0.77
M6	B11	2.54	2.74	1.08	2.21	0.87
M7	B12	2.43	2.51	1.03	2.64	1.09
M8	B13	2.30	2.47	1.07	2.66	1.16
M9	B14	2.66	2.95	1.11	2.93	1.10

Table 4.3: NoC router area results.

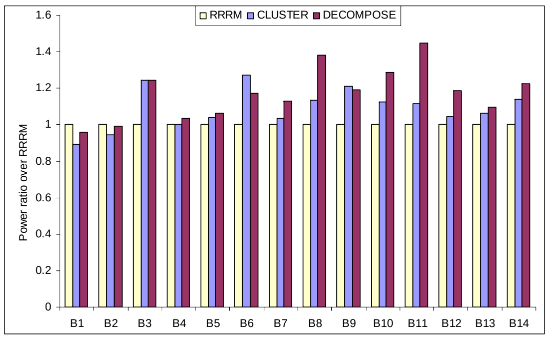
		RRRM	CLUSTER		DECOMPOSE	
application	label	Area(mm^2)	Area (mm^2)	Ratio /RRRM	Area (mm^2)	Ratio / RRRM
MMS	B1	0.98	0.95	0.97	0.92	0.97
V+M	B2	0.56	0.54	0.98	0.53	0.98
M+P	B3	0.22	0.31	1.39	0.31	1.00
V+M+M	B4	0.72	0.75	1.04	0.75	1.00
4in1	B5	0.78	0.86	1.10	0.86	1.00
M1	B6	0.90	1.14	1.27	1.20	1.06
M2	B7	1.56	1.59	1.02	1.61	1.01
M3	B8	1.78	2.03	1.14	2.03	1.00
M4	B9	1.93	2.68	1.39	2.41	0.90
M5	B10	2.68	3.06	1.14	3.01	0.98
M6	B11	2.93	3.56	1.21	3.36	0.94
M7	B12	3.95	3.92	0.99	4.24	1.08
M8	B13	4.90	4.81	0.98	5.19	1.08
M9	B14	5.05	5.54	1.10	5.61	1.01

can on average achieve a 3% reduction in average hop counts over CLUSTER and a 7% reduction in average hop counts over DECOMPOSE.

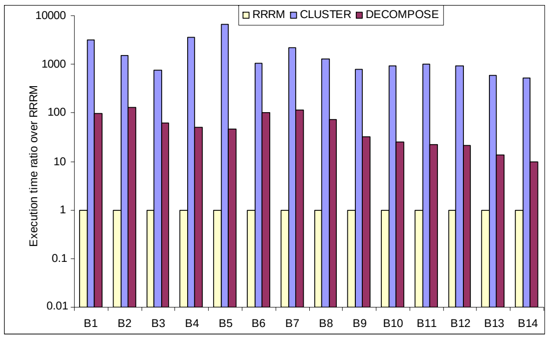
Finally, to evaluate the area costs of the synthesized solutions, we also used Orion [31, 32] to estimate the areas of the routers in the synthesized architectures, using the same 70nm technology used for power estimation. The area cost of a solution corresponds to the sum of the router areas in the solution. The results are presented in Table 4.3 and their relative results over RRRM are compared in Figure 4.4(d). Total area costs of all solutions produced by RRRM are better than those produced by CLUSTER and DECOMPOSE. In particular, on average, total area costs produced by RRRM are 12% better than those of CLUSTER and 1% better than those of DECOMPOSE.

Acknowledgement

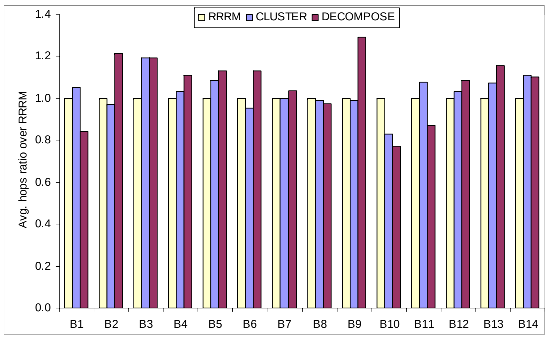
This chapter is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Joint Multicast Routing and Network Design Optimization for Networks-on-Chip”, *IET Computers and Digital Techniques*, accepted for publication, 2009. The dissertation author was the primary author of the above paper.



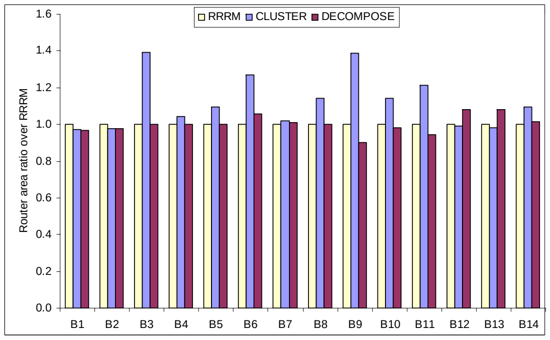
(a) Power



(b) Execution time



(c) Hop count



(d) Area

Figure 4.4: Comparisons of all algorithms relative to RRRM.

Chapter 5

3D Application-Specific NoC Architecture Synthesis

In this chapter, we present extensions to our proposed NoC synthesis algorithms to support 3D chip designs. In particular, we propose extensions to the rip-up and reroute algorithms presented in Chapter 4 because they generally produced better results with less CPU times than the algorithms presented in Chapter 3 based on set partitions and Steiner trees.

The rest of this chapter is organized as follows. Section 5.1 provides an overview of the 3D NoC design problem. Section 5.2 presents thaccurate power and delay models for 3D wiring and routers. Section 5.3 describes our 3D synthesis algorithm RRRM-3D. Finally, Section 5.4 presents experimental results.

5.1 Overview

The advent and increasing viability of 3D silicon integration technology have opened a new horizon for new on-chip interconnect design innovations. In particular, there has been considerable discussion in recent years on the benefits of three dimensional (3D) silicon integration in which multiple device layers are stacked on top of each other with direct vertical interconnects tunneling through them using through-silicon vias [83, 84, 85, 86] (Fig. 5.1). 3D integration promises to address many of the key challenges that arise from the semiconductor industry's relentless push into the deep nano-scale

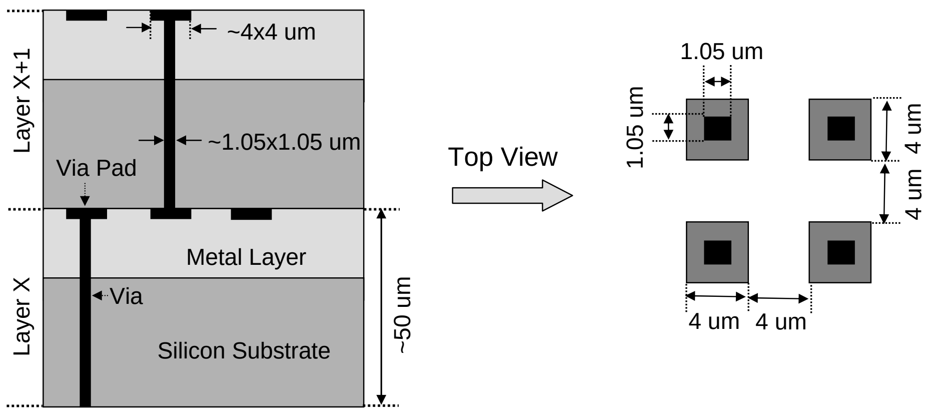


Figure 5.1: 3D silicon integration [100].

regime. First, as feature sizes continue to shrink, and integration densities continue to increase, interconnect delays have become a critical bottleneck in chip performance. By providing a third dimension of interconnect, wire delays can be substantially reduced by enabling greater spatial locality. Second, for many high-performance applications, such as video or graphics processing, the performance bottleneck is often in the chip-to-chip or chip-to-memory communication. Three dimensional integration offers the compelling advantage that massive amounts of bandwidth can be provided between device layers without incurring the usual latency penalty, leading potentially to new architectures that can achieve much higher performance. Third, fabrication technologies specific to functions such as RF circuits, memories, or optoelectronic devices are often incompatible with the processing steps needed for high performance logic devices. Three dimensional interconnect provides a flexible way to integrate these disparate technologies into a single systems-on-chip (SoC) design. Recent advances in 3D technology in the area of heat dissipation and micro-cooling mechanisms have alleviated earlier thermal viability and reliability concerns regarding stacked device layers.

In this chapter, we investigate the problem of designing application-specific 3D-NoC architectures for custom SoC designs. NoC design in 3D chips imposes new constraints and opportunities compared to that of a 2D NoC design. Current literature has focussed on regular 3D mesh NoC architectures [98, 99, 100], which is appropriate for regular 3D processor designs [88, 89, 91]. However, in the case of designing application-specific 3D NoC architectures for custom SoC designs, there are many choices that depend on the 3D floorplanning of cores, traffic requirements, and accurate power and delay models for 3D wiring. In our work, we derive accurate power models for 3D interconnects and routers. Then we extend our Ripup-Reroute-and-Router-Merging (RRRM) algorithm into the 3rd dimension to optimize topologies for 3D applications. Our 3D NoC design flow integrates 3D floorplanning and our synthesis process is both performance and power consumption aware.

5.2 3D Design Models

Power dissipation is a critical issue in 3D circuits due to the increased power density of stacked ICs and the low conductivity of the dielectric layers between the device layers. Therefore, designing custom 3D-NoC topologies that offer low power characteristics is of significant interests.

The different power consumption components that are comprised in 3D-NoC topologies are routers, horizontal interconnects that connect modules in the same 2D layer, and the through-silicon vias (TSVs) that connect modules or horizontal interconnects on different layers.

We will discuss the details of modelling these components in the following sections.

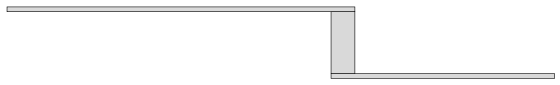
5.2.1 3D Interconnect Modelling

In 3D-NoCs, interconnect design imposes new constraints and opportunities compared to that of 2D NoC designs. There is an inherent asymmetry in the delay and power costs in a 3D architecture between the vertical and the horizontal interconnects due to differences in wire lengths. The vertical TSVs are usually few tens of μm in length whereas the horizontal interconnects can be thousands of μm in length. Consequently, extending a traditional 2D NoC fabric to the third dimension by simply adding routers at each layer and connecting them using vertical vias is not a good option, as router latencies may dominate the fast vertical interconnect. Hence, we explore an alternate option: a 3D interconnect structure that connects modules on different layers as shown in Fig. 5.2(a) and we derive an accurate model for it.

As discussed in Chapter 2, the target clock frequency is provided to our 3D-NoC synthesis design flow as a design parameter. However, depending on the network topology, long interconnects may be required to implement network links between routers, which may have wire delays that are larger than the target clock frequency. To achieve the target frequency, repeaters may need to be inserted. In the 2D design problem, interconnects can be modelled as distributed RC wires. One way to optimize the interconnect delay is to evenly divide the interconnect into k segments with repeaters inserted between them that are s times as large as a minimum-sized repeater. When minimizing power consumption is the objective, the optimum size s_{opt} and number k_{opt} of repeaters that minimize power consumption while satisfying the delay constraint can be determined for the interconnect [33]. For the 3D interconnect structure, we extended this distributed RC model. As shown in Fig. 5.2(b), a 3D interconnect is divided into k segments by repeaters. Among the k segments, $k-1$ segments are part of the horizontal interconnect with the same structure. The other one is a different structure with two horizontal parts connected by a vertical via. The delay and power consumption per bit of this interconnect can be modelled using the Elmore model, as in [33, 34, 98]. In order to take the vertical via into account for the delay and power calculation of the entire interconnect, we first consider the interconnect as k segments with the same structure. We use the methodology described in [33] to find s_{opt} and k_{opt} for an interconnect with specific length to minimize power while satisfying the delay constraint[1]. After that, the delay and power of each segment are known. Given the fixed length and the physical parameters of the via, the detailed structure of the segment including the via which gives the same delay as the delay of the original segment without via can be determined by properly choosing the length of the horizontal wire parts in this segment. Finally, the total length of the 3D interconnect can be adjusted to the original length by evenly adjusting the length of each segment.

Besides deciding the segment structure with vertical via, the via position on the interconnect also needs to be determined. That is, which wire segment is selected to include the via. As an example, in order to determine the influence of via positioning on the delay and power of the entire 3D interconnect, we performed experiments to evaluate the delay and power of an $8mm$ 3D interconnect with a via length of $150\mu m$ under different via positions. In the experiments, the physical and electrical parameters in $70nm$ technology are used and are listed in Table 5.1. The horizontal wires are implemented on the global

[1]Since inserting TSV adds delay, we tighten the delay constraints by some extent to get valid solutions.


(a) 3D interconnect.

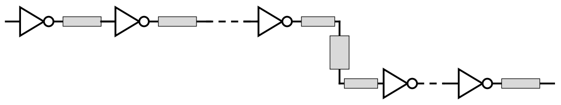

(b) Distributed RC model with repeaters

Figure 5.2: 3D interconnect model.

metal layers and their parameters are extracted from IRTS [2]. The parameters of vertical vias are obtained from [98]. For the vertical via, the length of $50\mu m$ is assumed for a via that connects adjacent layers.

For a 3D interconnect of $8mm$ in length, if the target frequency is 1GHz, then the power optimum solution using the methodology described in [33] is to divide the interconnect into 3 segments. Thus, there are 3 possible via positions with 3 interconnect structures correspondingly, which are shown in Fig. 5.3. The optimization result of each structure together with the result of the interconnect without vertical via (labelled as 2D-wire) are shown in Table 5.2. The differences of delay and power results of all structures relative to the 2D-wire results are also listed. The results show that the influence of the vertical via on the total delay and power consumption of the entire interconnect is very small. The $150\mu m$ via results in 0.25% increase in delay and 2.85% increase in power over the $8mm$ interconnect. The results also show that the position of via on the interconnect has little effect on the delay and power. All the structures result in the same total delay and power. Thus, for our 3D-NoC synthesis algorithm, we can safely choose to position the via in the first segment of the interconnect for all 3D interconnects in the synthesized NoC topology for the purpose of computing the interconnect power costs.

In our 3D-NoC synthesis design, we use the above 3D interconnect model to evaluate optimum power consumption of interconnects with different wire lengths under the given design frequency and delay constraint. These results are provided to the design flow in the form of a library. We emphasize that the focus of this work is on 3D-NoC synthesis algorithms. We readily admit that 3D interconnect optimization is a complex problem and a subject of separate research. New or alternative 3D interconnect models can be easily used with our synthesis algorithms and design flow.

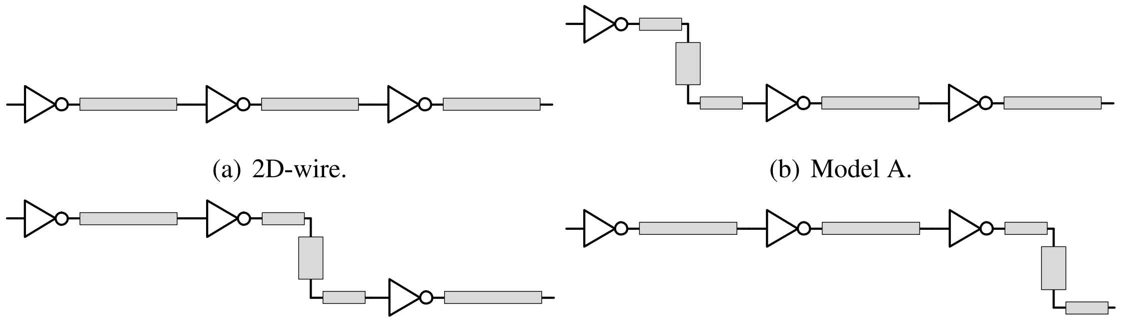

(a) 2D-wire. (b) Model A. (c) Model B. (d) Model C.

Figure 5.3: Different structures for an 8mm 3D interconnect.

Table 5.1: Interconnect Parameters

Interconnect Structure	Parameter			
	Electrical		Physical	
Horizontal Bus	$\rho = 2.53\mu\Omega \cdot cm$	$k_{ILD} = 2.7$	$w = 500nm$	$s = 500nm$
	$r_h = 46\Omega/mm$	$c_h = 192.5fF/mm$	$t = 1100nm$	$h = 800nm$
Vertical Bus	$\rho = 5.65\mu\Omega \cdot cm$	$r_v = 51.2\Omega/mm$	$w = 1050nm$	$L_{via} = 50\mu m$
	$c_v = 600fF/mm$			

Table 5.2: Power and delay comparison of 3D interconnect models

Model	Power		Delay	
	(mW)	% diff to 2D-wire	(ns)	% diff to 2D-wire
2D-wire	0.3909	0.00%	0.1951	0.00%
A	0.4020	2.85%	0.1956	0.25%
B	0.4020	2.85%	0.1956	0.25%
C	0.4020	2.85%	0.1956	0.25%

5.2.2 Modelling Routers

To evaluate the power of the routers in the synthesized NoC architecture, we extended the router power model in 2 dimensions to 3 dimensions. The routers are still located on a 2D layer. The ports of routers on the same layer are connected by horizontal interconnects whereas the ports of routers on different layers are connected by 3D interconnects. We again use the NoC power-performance simulator Orion [31, 32] to estimate the detailed power characteristics for different power components of a router for different input/output port configurations. The power per bit values are also used as the basis for the entire router power estimation under different configurations.

5.3 Design Algorithms

We extend our Ripup-Reroute-and-Router-Merging (RRRM) algorithm into 3D (RRRM-3D). As discussed in Chapter 4.2.2, the entire process of RRRM-3D is decomposed into the inter-related steps of constructing an initial network topology, rip-up and rerouting flows to design the network topology, inserting the corresponding network links and router ports to implement the routing, and merging routers to optimize network topology based on design objectives.

The RIPUP-REROUTE-3D algorithm for routing flows and the ROUTER-MERGING-3D algorithm to optimize topologies are based on using the above proposed power models of 3D network links and router ports as cost evaluation criteria. The details of the algorithm are similar to RRRM, which is discussed in Algorithm 6-8 and Algorithm 5. They are not repeated here.

5.4 Experimental Results

5.4.1 Experimental Setup

We have implemented our proposed algorithm RRRM-3D in C++. In our experiment, we aim to evaluate the performance of our proposed algorithm RRRM-3D on benchmarks with the objective of minimizing the total power consumption of the synthesized

NoC architectures under the specific performance constraint for the traffic flows. The performance constraint is specified in the form of average hop counts of all the traffic flows in the benchmarks. The total power consumption includes both the leakage power and the dynamic switching power of all network components. We use the same power-performance simulator Orion and applied the same design parameters as described in Chapter 3.

All existing published benchmarks are targeted to 2D architectures. However, their sizes are not big enough to take advantage of 3D network topologies. In the absence of published 3D benchmarks with a large number of available cores and traffic flows, we generated a set of synthetic benchmarks by extending the NoC-centric bandwidth-version of Rent's rule proposed by Greenfield et al. [46]. They showed that the traffic distribution models of NoC applications should follow a similar Rent's rule distribution as in conventional VLSI netlists. We used this NoC-centric Rent's rule [46, 47] to generate large 3D NoC benchmarks for 3D circuits with varying number of cores in each layer and flows of varying data rate distributions. The benchmarks are generated for 3D circuits with 3 layers and 4 layers respectively with face-to-back bounding between each layer. The total number of cores for these benchmarks are ranging from 48 to 120. The total number of flows are ranging from 101 to 280.

Our work is among the first in the area of application-specific NoC synthesis of 3D network topologies. In the absence of previously published work on this area, direct comparison with others' work is unavailable. To evaluate the effectiveness of our proposed algorithm, we have generated a full 3D mesh implementation for each benchmark for comparisons. In a full 3D mesh implementation, each module is connected to a router with 7 input/output ports, with 1 local port, 4 ports connecting to the four directions in the same layer, and 2 ports connecting to the upper and lower adjacent layers. Packets are routed using XYZ routing over the mesh from source to destination. We also generated a variant of the basic mesh topology called optimized mesh (opt-mesh) by eliminating router ports and links that are not used by the traffic flows.

All experimental results were obtained on a 1.5GHz Intel P4 processor machine with 512MB memory running Linux.

Table 5.3: 3D NoC synthesis results.

				RRRM						mesh		opt-mesh	
Bench.	|L|	|cores|	|flows|	Power (W)	Ratio to mesh	Ratio to opt-mesh	Avg. Hops	Ratio to mesh/opt	Time (sec)	Power (W)	Avg. Hops	Power (W)	Avg. Hops
B1	3	48	101	0.497	0.25	0.50	2.48	0.98	4	1.951	2.54	0.990	2.54
B2	3	60	133	0.654	0.26	0.49	2.73	0.99	15	2.530	2.77	1.326	2.77
B3	4	64	149	0.788	0.28	0.46	2.05	0.60	37	2.769	3.40	1.712	3.40
B4	3	75	169	0.850	0.26	0.51	2.38	0.88	79	3.269	2.69	1.678	2.69
B5	4	80	177	0.867	0.24	0.53	2.25	0.82	137	3.550	2.75	1.637	2.75
B6	3	90	203	1.089	0.27	0.48	2.65	0.85	235	4.032	3.11	2.254	3.11
B7	4	100	228	1.083	0.24	0.44	2.68	0.88	443	4.599	3.06	2.461	3.06
B8	3	108	248	1.335	0.27	0.45	2.22	0.63	446	4.975	3.52	2.979	3.52
B9	4	120	280	1.426	0.25	0.44	2.59	0.81	1144	5.670	3.21	3.233	3.21

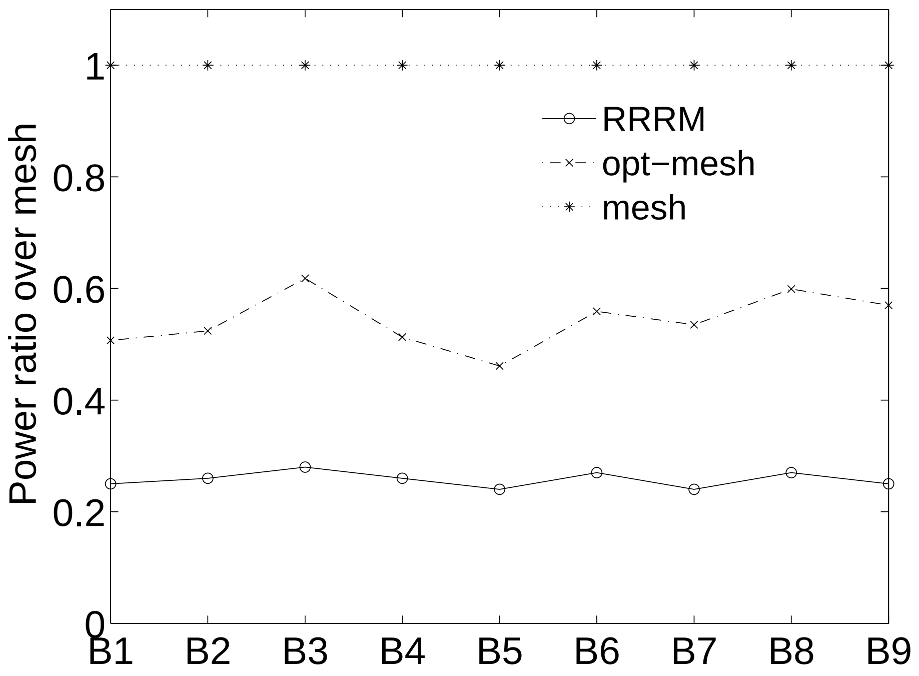
5.4.2 Comparison of results

The synthesis results of our algorithm on all benchmarks at 70nm with comparison to results using mesh and opt-mesh topologies are shown in Table 5.3. For each algorithm, the power results and the average hop counts are reported.

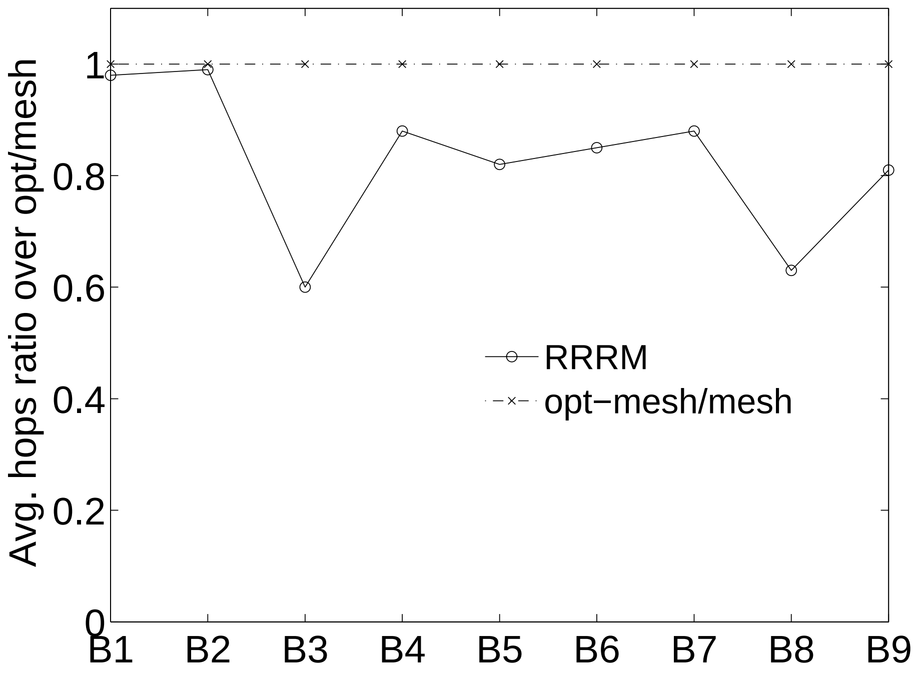
In the experiments, we used the average hop count results of 3D mesh topologies as the performance constraints feeding to RRRM-3D for each benchmark. The average hop count results for the different benchmarks of 3D mesh topologies reported in Table 5.3 are small, all under 3.5. The average hop count results of RRRM-3D on all benchmarks relative to opt-mesh/mesh implementation are graphically compared in Fig. 5.4(b). The results show that the results of RRRM-3D satisfies constraints for all the benchmarks. On average, RRRM-3D can achieve 17% average hop count reduction over the mesh topology.

The power consumption results of RRRM-3D and opt-mesh relative to mesh implementations are graphically compared in Fig. 5.4(a). The results show that RRRM-3D can efficiently synthesize NoC architectures that minimize power consumption under the performance constraint. It can achieve substantial reduction in power consumption over the standard mesh and opt-mesh topologies in all cases. In particular, it can achieve on average a 74% reduction in power consumption over standard mesh topologies and a 52% reduction over the optimized mesh topologies.

The execution times of RRRM are also reported in Table 5.3. The results show that RRRM works very fast. For the largest benchmarks with 120 cores and 280 flows, it can finish within 20 minutes.



(a) Power



(b) Hop count

Figure 5.4: Comparisons of all algorithms on benchmarks.

Acknowledgement

This chapter is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Design of Application-Specific 3D Network-on-Chip Architectures”, *The International Conference on Computer Design (ICCD 2008)*, 2008. The dissertation author was the primary author of the above paper.

Chapter 6

Design of Application-Specific NoC for Multiple Usage Scenarios

6.1 Overview

Increasingly, multiprocessor SoCs are designed to support different usage scenarios since such SoC designs may be employed in different products or in products with different operation modes. For these embedded applications, the NoC must be designed to satisfy the communication characteristics and performance constraints of *all* usage scenarios considered. Although a regular full-mesh architecture is able to support multiple traffic profiles, custom NoCs designs tailored and optimized for a given set of traffic profiles can significantly outperform a regular mesh architecture in terms of power, performance, and area. Prior work has either only considered a *single* traffic profile in their custom NoC synthesis formulations [24, 29, 22, 23, 25, 28, 57, 58] or has assumed a *given* or *regular* network topology for supporting different usage scenarios [59, 60].

In this chapter, we present extensions to the RRRM algorithms presented in Chapter 4 to synthesize custom NoC architectures that are optimized for a given set of traffic profiles. These traffic profiles correspond to the communication requirements for different usage scenarios.

6.2 Problem and Formulation

The composing parts of the input specification to our design flow is similar as ones discussed in previous chapters: the first part is a set of application specifications, where each application specification specifies a list of tasks and the corresponding communication characteristics and performance constraints between them. The usage scenarios for these applications are also provided, which describe which applications may be running concurrently. The second part is the floorplan for the modules that will be used to implement the application tasks. These modules can correspond to a variety of programmable processors and intellectual property (IP) cores. In particular, the floorplan specifies the locations of these modules. Finally, the third part of the input specification provides the task-to-module mapping[1].

Given the communication requirements between tasks, the usage scenarios, the floorplan, and the task-to-module mapping, a set of traffic profiles between modules can be derived that specifies the inter-module communication demands. Each of these derived traffic profile is captured in a corresponding graph representation called a *Communication Demand Graph* (CDG), which is defined in Chapter 2.
For N usage scenarios, we have N CDGs, $G_1, G_2, \ldots, G_N$.

Based on the optimization goals and cost functions specified by the user, the output of our NoC synthesis problem is an optimized custom network topology with *pre-determined routes* for the specified traffic flows of each application on the network such that the data rate requirements for each usage scenario are satisfied.

As usual, to obtain the best solutions with minimum power consumption, accurate power models for interconnects and routers are derived. They are provided to the synthesis design flow as a library and utilized by the synthesis algorithms as evaluation criteria. The application-specific NoC synthesis problem for multiple traffic profiles can be formulated as follows:

Input:

- The N communication demand graphs $G_1(V_1, E_1, \pi_1, \lambda_1)$, . . ., $G_N(V_N, E_N, \pi_N, \lambda_N)$ for N usage scenarios.

[1]Often the task-to-module mapping is determined by the designer based on knowledge about the design and/or based on the specific hardware capabilities of particular processors or IP cores and the specific required functionalities of the different tasks.

- The NoC network component library $\Phi(I, J)$, where I provides the power and area models of routers with different sizes, and J provides power models of physical links with different lengths.
- The target clock frequency, which determines the delay constraint for links between routers.
- The hop count constraints $H_1, \ldots, H_N$ for N usage scenarios.

Please note that the last three elements of the input are the same as the ones for a single usage scenario NoC synthesis problem.

Output:

- A NoC architecture $T(R, L, C)$, where R denotes the set of routers in the synthesized architecture, L represents the set of links between routers, and a function $C : V \rightarrow R$ that represents the connectivity of a processor to a router.
- Sets of ordered paths $P_1, P_2, \ldots, P_N$, where each $p_{ij} \in P_s = (r_i, r_j, \ldots, r_k)$, $r_i, \ldots, r_k \in R$, represents a route for a traffic flow $e(v_i, v_k) \in E_s$ for usage scenario G_s.

Objective:

- Power minimization for the synthesized NoC architecture.

6.3 Multi-Profile Network Design Algorithms

We present in this section our custom NoC synthesis algorithm for supporting multiple usage scenarios. The algorithm is based on the Ripup-Reroute-and-Router-Merging (RRRM) procedure that was described in Chapter 4 for the single traffic profile case. To support multiple usage scenarios, we present a multi-traffic-profile version of RRRM called Multi-Profile-RRRM (MPR).

6.3.1 Initial network construction

The details of MPR are described in Algorithm 9. MPR takes a set of CDGs, an evaluation function, and the hop count constraint of each usage scenario as inputs, and it generates an optimized network architecture as output. It starts with initializing a network topology by a simple router allocation and flow routing scheme. It then uses a procedure of rip-up and rerouting of flows to refine and optimize the network topology. After that, a router merging step is performed to further optimize the topology to obtain the best result. In order to support all usage scenarios, we consider all flows from all N CDGs in the rip-up and reroute procedure, and we design a network that can support *any* of the N CDGs. For this, we introduce a notion of *exclusive flows*.

Definition 5 (Exclusive Flows)**.** *Two flows f_1 and f_2 are said to be exclusive, denoted as $f_1\#f_2$, if they cannot simultaneously occur.*

Definition 6 (Exclusive Flow Sets)**.** *Two disjoint sets F_1 and F_2 are said to be exclusive if*

$$\forall f_1 \in F_1, \forall f_2 \in F_2, f_1\#f_2$$

Flows from two different traffic profiles form a disjoint exclusive flow sets.

The initial router allocation steps of the MPR procedure are the same as the ones of the RRRM. After an initial router allocation, a Routing Cost Graph (RCG)(Which is defined in Chapter 4) is generated (Line 2).

Following the initial router allocation, an initial network topology is constructed using the InitialNetworkConstruction() procedure. Each flow $e_k = (s_k, d_k)$ in each CDG is routed using a direct connection from router r_{s_k} to router r_{d_k}, where r_i is the router that core i connects to, and the path is saved in $path(e_k)$. The links and router ports are configured and saved. The flows that use such links and router ports are updated. A set of exclusive flow sets are associated with each link and router port. Flows from different applications are saved in different *exclusive flow sets*. Flows from the same application are in the same flow set. The bandwidth contributions from different traffic profiles are updated separately for links and routers as well.

6.3.2 Flow Ripup and Rerouting

Once the initial network is constructed and the initial flow routing is done, the key procedure of the algorithm, MP-RIPUP-REROUTE, is invoked to route the flows and to find an optimized network topology. The details of MP-RIPUP-REROUTE are described

Algorithm 9 MPR(G, C, L, H)

```
Input: $G = \{G_1, G_2, \ldots, G_N\}$: CDGs, $C$: cost function,
    $L$: library of network components,
    $H = \{H_1, H_2, \ldots, H_N\}$: hop count constraints
Output: $T$: synthesized network topology
1: $R$ = InitialRouterAllocation($G$)
2: $RCG$ = ConstructFullyConnectedGraph($R$)
3: $(links, routers)$ = InitialNetworkConstruction($G, RCG$)
4: $(links, routers)$ = MP-RIPUP-REROUTE($G$,$RCG$, $C$, $L$, $H$)
5: $cost$ = EvaluatePowerConsumption($links, routers$)
6: RouterMerging($cost, links, routers$)
7: $T$ = ObtainBestTopology($links, routers$)
8: return $T$
```

```
InitialNetworkConstruction($G, RCG$)
1: for all CDG $G_i \in G$ do
2:     for all flow $e_k = (s_k \rightarrow d_k) \in E_i$ do
3:         route $e_k$ using a direct connection between $r_{s_k}$ and $r_{d_k}$
4:         update exclusive flow sets routed on $r_{s_k}$, $r_{d_k}$ and $link(r_{s_k}, r_{d_k})$
5:         update bandwidths of routers $r_{s_k}, r_{d_{ki}}, link(r_{s_k}, r_{d_{ki}})$
6:         $path(e_k) = (r_{s_k} \rightarrow r_{d_k})$
7:         if $link(r_{s_k}, r_{d_k})$ not meet delay constraint, update $wgt(r_{s_k}, r_{d_k}) = \infty$ in $RCG$
8:     end for
9: end for
```

in Algorithm 10. In the MP-RIPUP-REROUTE procedure, given the hop count constraints of each application, each flow routing step is formulated as a performance constrained shortest path problem.

When a flow is ripped-up and rerouted, its current path is deleted and the links and router ports resources it occupies are released (Line 4). Then based on the current network connectivity and resources occupation the RCG related to this flow is built and the weights of all edges in RCG are updated (Line 5).

In order to support multiple traffic profiles, when calculating the weights of the edges, we separate the cost calculation into a *static* component, and a *dynamic* component. The static costs are caused by the installations of routers and links. They are related to the leakage power of routers and links. As long as a router port or a link needs to be installed to support an application, The static cost is calculated as appropriate. The dynamic costs are caused by routing flows on the routers and links. They are related to the dynamic power of routers and links which are the functions of traffic data rates. For the dynamic component, the dynamic cost for each application (exclusive flow set) is computed separately. The *maximum* dynamic power cost among all applications is taken as the *worst-case* dynamic cost since only one application can run at a time. In order to do this, for each router port and link, the exclusive flow sets for flows routing through them are maintained and tracked. In particular, for every pair of routers in RCG, the cost of using those routers and the link connecting them is evaluated. This cost depends on the sizes of the routers, the traffic already routed on the routers and the connectivity of the routers to other routers. It also depends on whether an existing physical link will be used or a new physical link needs to be installed. If there are already router ports and links that can support the traffic, the marginal cost of reusing those resources is calculated. Otherwise, the cost of opening new router ports and installing new physical link to support the traffic is calculated. The cost is assigned as edge weight to the edge connecting the pair of routers in RCG. If the physical links used to connect the routers cannot satisfy the delay constraints, a weight of infinity is assigned to the corresponding edges in RCG.

Once the weights are assigned to all edges, the least cost path that satisfy the hop counts constraint of the flow is chosen for routing that flow using the *H-hops shortest paths* algorithm proposed in [61]. The H-hops shortest paths problem is defined as follow.

Definition 7 (H-hops Shortest Paths Problem)**.** *For a given graph $G(N, E)$, a source node $s \in N$, and a maximal hop count H, find the least cost h-hop path from s to every destination node $u \in N$, for each hop count value h, $1 \leq h \leq H$.*

After the path is determined, the routers and links on the chosen path are updated.

This MP-RIPUP-REROUTE process is repeated for all flows. The result of this procedure depends on the order that the traffic profiles and flows are considered. In our implementation, we allow users to define the priority of the applications so that the applications with higher priority can be rip-up and rerouted before the applications with lower priority. And the entire procedure can be repeated for several times for each application to reduce the dependency of the results on flow ordering. Once the path of each flow is decided, the size of each router, the links that connect the routers are determined. Routers that have no traffic multiplexing or de-multiplexing are deleted and links are reconnected.

The remaining routers and links constitute the network topology. The total implementation cost of all the routers and links in this topology is evaluated and the network topology is obtained.

Algorithm 10 MP-RIPUP-REROUTE(G, RCG, C, L, H)

```
Input: $G = \{G_1, G_2, \ldots, G_N\}$: CDGs, $RCG$: router cost graph, $C$: cost function, $L$: library of network components,
    $H = \{H_1, H_2, \ldots, H_N\}$: hop count constraints
Output: $links, routers$: links and routers of synthesized network
1: while need another round do
2:   for all $G_i \in G$ in descending order of the priority do
3:     for all flow $e_k \in E_i$ in increasing order of $\lambda(e_k)$ do
4:       delete $path(e_k)$ and release the link and router resources it occupied
5:       update exclusive flow sets on the link and router ports
6:       update all edge weights in RCG for flow $e_k$, according to power consumption of the corresponding links and routers resources
7:       ($P^1(s_k, d_k)$,$P^2(s_k, d_k)$,. . .,$P^{H_i}(s_k, d_k)$) = H-hops-ShortestPaths($G_i, s_k, d_k, H_i$)
8:       $path(e_k)$ = least-cost path in ($P^1(s_k, d_k)$, $P^2(s_k, d_k)$,. . .,$P^{H_i}(s_k, d_k)$)
9:       Update $link, BW\_avail, routers$ for $path(e_k)$ and update exclusive flow sets on them
10:    end for
11:  end for
12: end while
13: return $links, routers$
```

Finally, after the physical network topology has been generated using MP-RIPUP-REROUTE, a greedy router merging step is used to further optimize the topology to reduce the power consumption.

6.4 Evaluation

6.4.1 Experimental Setup

We have implemented our proposed algorithm MPR in C++. In all our experiments, we aim to evaluate the performance of MPR on all benchmarks with the objective of minimizing the total power consumption of the synthesized NoC architectures under the performance constraints to support multiple traffic profiles. The total power consumption

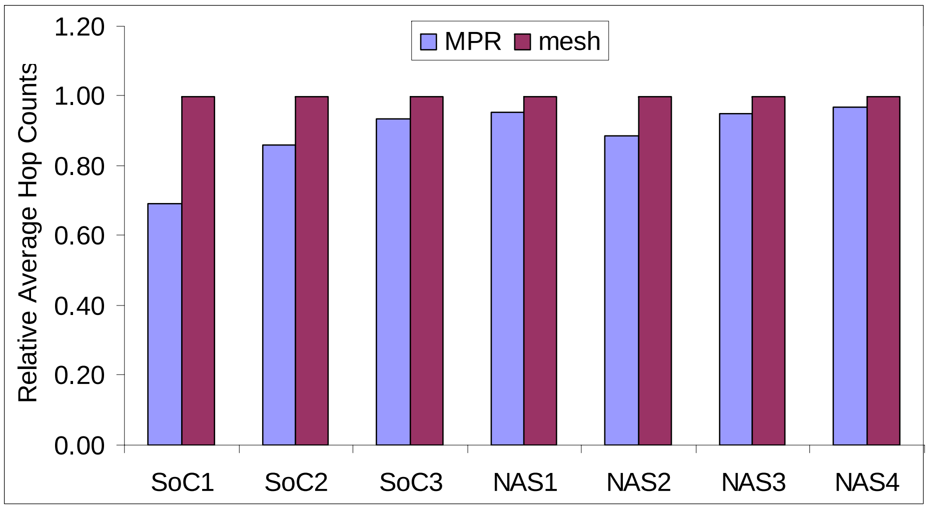


Figure 6.1: Hop count comparisons of MPR vs. regular mesh on benchmarks.

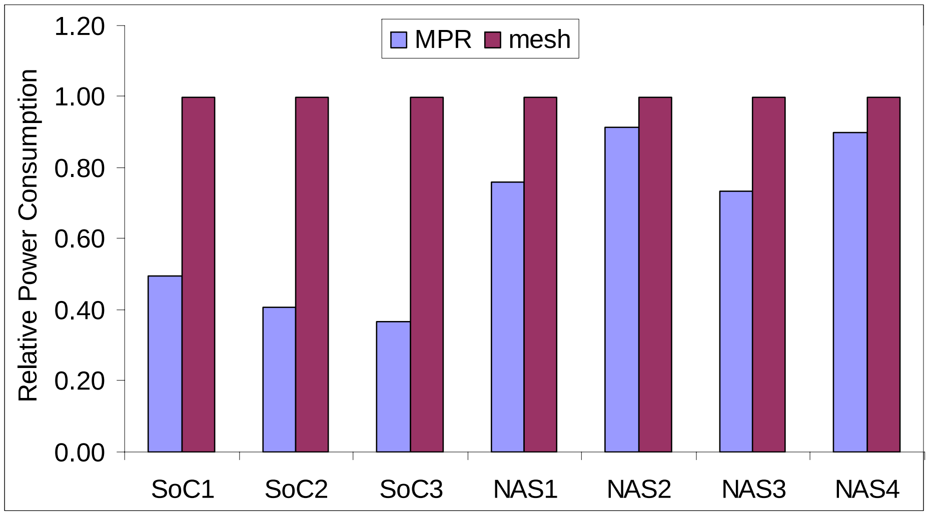


Figure 6.2: Power comparisons of MPR vs. regular mesh on benchmarks.

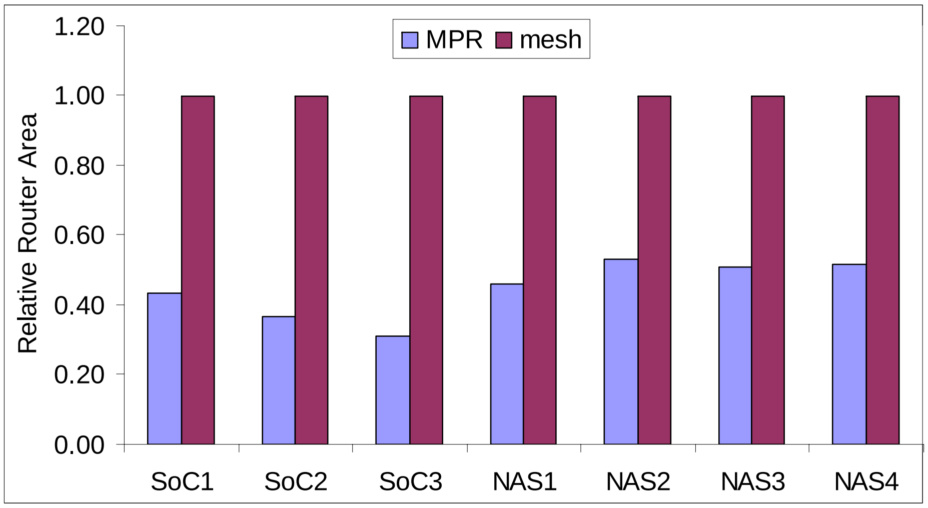


Figure 6.3: Area comparisons of MPR vs. regular mesh on benchmarks.

includes both the leakage power and the dynamic switching power of all network components. We use the same power-performance simulator Orion and applied the same design parameters as described in Chapter 2.

Table 6.1: Characteristics for Five NAS Parallel Benchmarks

Benchmark	Label	Class	Size	# Proc	# comm	Label	Class	Size	# Proc	# comm
block tridiagonal solver (BT)	BT-A-16	A	64^3	16	96	BT-A-36	A	64^3	36	216
Conjugate gradient (CG)	CG-A-16	A	14000	16	48	CG-A-36	A	14000	36	128
LU solver (LU)	LU-A-16	A	64^3	16	48	LU-A-36	A	64^3	36	104
Multigrid (MG)	MG-A-16	A	256^3	16	72	MG-A-36	A	256^3	36	176
Pentadiagonal solver (SP)	SP-A-16	A	64^3	16	96	SP-A-36	A	64^3	36	216
BT	BT-B-64	B	102^3	64	384	BT-B-121	B	102^3	121	726
CG	CG-B-64	B	75000	64	256	CG-B-128	B	75000	128	640
LU	LU-B-64	B	102^3	64	224	LU-B-128	B	102^3	128	466
MG	MG-B-64	B	256^3	64	408	MG-B-128	B	256^3	128	920
SP	SP-B-64	B	102^3	64	384	SP-B-121	B	102^3	121	720

Two sets of benchmarks were used to evaluate the proposed algorithm. The first set of benchmarks are real SoC applications used beforeWe group those applications into three multiple traffic profile groups according to their sizes, each consisting of several applications. The second set of benchmarks are NAS parallel benchmarks [62]. We ran the class A benchmarks with 16 and 36 processes and class B benchmarks with 64 and 121 processes (for BT and SP) or 128 processes (for other six benchmarks). The traffic patterns were then extracted using Intel Trace Analyzer and Collector 7.1 on a Linux platform [63]. Among the eight benchmarks, we excluded EP, FT and IS in our experiments because there were too few communications among the processes. Table 6.1 lists the characteristics of the

Table 6.2: Multiple Traffic Profile Benchmarks

Bench group	# Bench	Benchmarks	# Processors
SoC1	5	VOPD, MPEG4,MWD,G5,MMdec	12
SoC2	5	PIP, H263, MP3enc, MP3dec, H263dec	8
SoC3	5	MMS, VOPD,MPEG4,MWD,PIP	25
NAS1	5	x-A-16	16
NAS2	5	x-A-36	36
NAS3	5	x-B-64	64
NAS4	5	x-B-121,x-B-128	128

rest five benchmarks that were used in the experiments with different class type and number of processes configurations. Then for each number of processes, we grouped all five benchmarks together into multiple traffic profile benchmarks. All the multiple traffic profile benchmarks used in our experiments from these two sets are summarized in Table 6.2.

For the two sets of benchmarks, we mapped each benchmark onto a chip-multiprocessor architecture where the number of processors is equal to the maximum number of tasks that a usage scenario for that benchmark will require. These processors are arranged into a two-dimensional rectangular floorplan. For the task-to-module mapping, we implemented a simulated annealing based algorithm for deciding on the assignment of tasks to processors. We employed a simple metric for modeling the communicating costs between communicating processors based on the bandwidth requirements of the corresponding flows, and we used this metric in the simulated annealing procedure to find a task-to-module mapping that minimizes the total communication cost. Then we derived the communication demand graphs for the different usage scenarios based on these task-to-module mappings, with the derived communication demand graphs serving as inputs to our NoC synthesis algorithms.

To evaluate the effectiveness of MPR, we generated a regular mesh implementation for comparison. We assumed the same rectangular processor floorplans and task-to-module mappings that were used in our custom NoC generations. In a mesh implementation, each processor is connected to a router with 5 input/output ports, with 1 local port and 4 ports connecting in four directions. Packets are routed using XY routing over the mesh from source to destination.

6.4.2 Comparison of Results

Table 6.3 shows the synthesis results of MPR on all multiple traffic profile benchmarks at 70nm with comparisons to results using mesh topologies. For each algorithm, the average hop counts, the power results, and the router areas of the synthesized NoC topologies of each benchmark group are reported.

Table 6.3: NoC synthesis results for multiple traffic profiles.

Benchmark group	MPR						mesh		
	Power (W)	Ratio to mesh	Hops	Ratio to mesh	Router Area (mm^2)	Ratio to mesh	Power (W)	Hops	Router Area (mm^2)
SoC1	0.148	0.49	2.03	0.69	0.87	0.43	0.299	2.94	2.01
SoC2	0.062	0.41	2.36	0.86	0.41	0.37	0.153	2.74	1.12
SoC3	0.245	0.37	2.36	0.93	1.53	0.31	0.669	2.53	4.93
NAS1	0.367	0.76	2.53	0.95	1.33	0.46	0.482	2.65	2.89
NAS2	1.193	0.91	3.05	0.89	3.98	0.53	1.305	3.44	7.50
NAS3	2.063	0.73	3.49	0.95	7.22	0.51	2.808	3.68	14.23
NAS4	13.322	0.90	4.51	0.97	23.33	0.52	14.844	4.66	45.09

In our experiments, we used the average hop count results of each traffic profile over a mesh topology as the performance constraints considered by MPR for each benchmark. The average hop count results of MPR on all benchmarks relative to mesh implementations are graphically compared in Fig. 6.1. The results show that the solutions produced by MPR are able to satisfy the provided performance constraints for all benchmarks. On average, MPR can achieve a 11% average hop count reduction over the mesh topology.

The power consumption results of MPR relative to mesh implementations are graphically compared in Fig. 6.2. The results show that MPR can efficiently synthesize NoC architectures that minimize power consumption under performances constraints to support multiple traffic profiles for both SoC and NAS-parallel benchmarks. It can achieve substantial reduction in power consumption over standard mesh topologies in all cases. In particular, it can achieve on average a 35% reduction in power consumption over standard mesh topologies for all benchmarks. It is worth mentioning that for the SoCs benchmark sets where each module only needs to communicate with a small number of modules, the application-specific NoC topologies are more appropriate for supporting multiple traffic profiles. On average, our algorithm can achieve a 58% reduction in power consumption over mesh topologies. Even for the communication-intensive parallel benchmarks in the NAS benchmark suite, an application-specific NoC architecture can be significantly better than a regular mesh. On average, our algorithm can achieve a 17% reduction in power consumption over mesh topologies on these NAS benchmarks.

To evaluate the area costs of the synthesized solutions, we also used Orion [31] to estimate the areas of the routers in the synthesized architectures, using the same 70nm technology used for power estimation. The area cost of a solution corresponds to the sum of the router areas in the solution. The results are presented in Table 6.3 and all the area results of MPR relative to mesh results are graphically compared in Fig. 6.3. In comparisons to the area costs of the mesh solutions, our algorithms are on average 55% lower than mesh.

Acknowledgement

This chapter is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Design of Application-Specific On-Chip Networks for Multiple Usage Scenarios", submitted to *IEEE Embedded Systems Letters*, 2009. The dissertation author was the primary author of the above paper.

Chapter 7

Deadlock-Free NoC Architecture Synthesis

Finally, in this chapter, we address deadlock considerations in our NoC synthesis algorithms. Deadlock-free routing is an important consideration for the correct operation of custom NoC architectures. The problem is very well studied and analyzed in the literature. In [66], Dally and Seitz proposed a necessary and sufficient condition for deterministic deadlock-free routing using the concept of a channel dependency graph. For general multiprocessor systems that can be programmed to run different applications, or in the case when adaptive routing is used, the problem is complicated by the challenge that the flows and routing paths are not necessarily known in advance [64, 65, 67]. On the other hand, for our custom NoC synthesis problem, the traffic flows and their required data rates are specified in advance, and *pre-determined* routes, for both unicast and multicast flows, are decided and fixed as part of the NoC synthesis process.

For our deterministic routing problem, deadlock-free operations can be ensured in the following ways:

7.1 Statically Scheduled Routing

For our NoC solutions, the required data rates are specified and the routes are fixed. In this setting, data transfers can be *statically scheduled* along the pre-determined paths with resource reservations to ensure deadlock-free routing. As advocated in [101], statically scheduled traffic is an effective option for providing *guaranteed traffic*. It has been shown in [101, 102] that router microarchitectures can be effectively extended so that statically scheduled traffic can co-mingle well with best effort traffic.

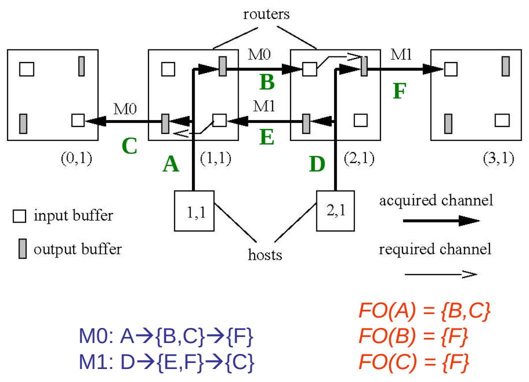
7.2 Virtual Channels

As shown in [66], a necessary and sufficient condition for deadlock-free routing is the absence of cycles in a channel dependency graph. In our problem setting, the traffic flows are known in advance, and the synthesis procedure is responsible for deciding on a good network topology and finding pre-determined routes for the specified traffic flows. Given that the traffic flows, routing paths, and network topology are all fixed by the synthesis procedure, we can construct a corresponding channel dependency graph where each node in the graph corresponds to a channel in the network, and a directed edge is added from c_i to c_j if there is a channel dependence from c_i to c_j (i.e., a flow is holding c_i and waiting for c_j).

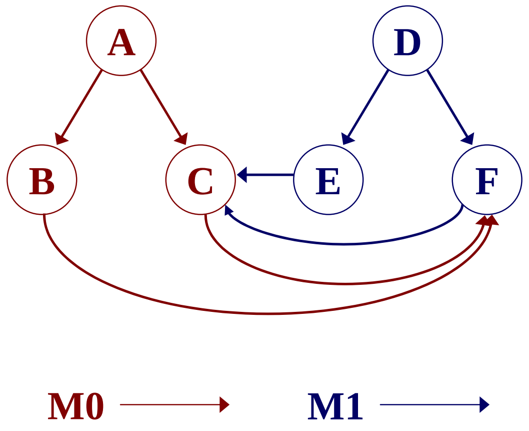
For the unicast case, the construction is straightforward: if c_j follows immediately after c_i in a routing path for a flow, then we add an edge from c_i to c_j. The multicast case is more complicated as deadlocks may be caused by the resource dependence between two multicast trees, even though the trees may not form a cycle topologically. To consider the multicast case as well, we propose and use an *extended* channel dependency graph construction as follows. If a multicast flow enters a router r through channel c_i and bifurcates from r to a group of channels $FO(c_i)$, then we add an edge from c_i to each $c_j \in FO(c_i)$. We refer to $FO(c_i)$ as the *fanout set* of c_i. It is defined as follows.

Definition 8. *A multicast flow enters a router r through channel c_i and bifurcates to a group of channels $FO(c_i)$, $FO(c_i)$ is the* fanout set *of c_i.*

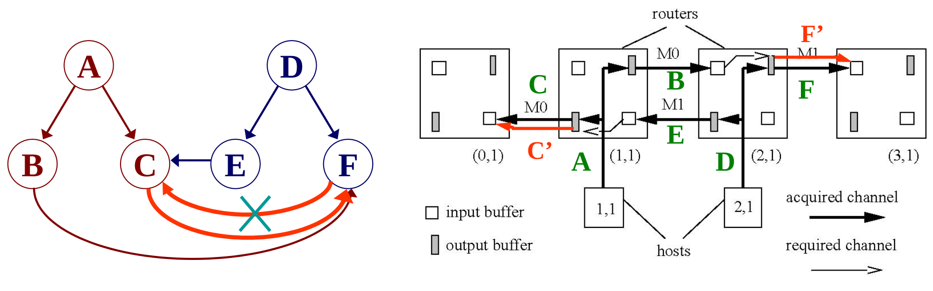
The intuition is that if a multicast flow has acquired channel c_i, then it can only proceed if it can acquire all the channels at the fanout set of c_i. In addition, let $c_j \in FO(c_i)$ and $c_k \in FO(c_i)$ be two channels in the fanout set of c_i. Then we also need to add an edge from c_j to each channel in the fanout set of c_k as well. That is, we need to add an edge from c_j to each $c_\ell \in FO(c_k)$. The intuition is that if a multicast flow has acquired channels c_j and c_k, it cannot proceed past c_j unless it can proceed past c_k as well. Similarly, we need



(a) Deadlock example.



(b) Extended channel dependency graph.



(c) Virtual channel insertion.

Figure 7.1: Illustration of the virtual channel insertion procedure.

to add an edge from c_k to each channel in the fanout set of c_j (i.e., an edge from c_k to each $c_m \in FO(c_j)$). This extended construction treats unicast flows as a special case.

Figure 7.1 shows the example of how to use virtual channel to avoid deadlock. Figure 7.1(a) shows part of a network with four routers. There are two multicast flows $M0$ and $M1$. $M0$ is from host (1,1) through channel A to B, C then to F. $M1$ is from host $(2, 1)$ through channel D to E, F then to C. The fanout set of channel A is B, C. The fanout set of channel B and C is F. In the current state, there is a deadlock situation since $M0$ has acquired channel A and B, C but needs F to proceed. $M1$ has acquired channel D and E, F but needs C to proceed. Both need resource that is hold by others. The extended channel dependency graph is shown in Figure 7.1(b).

Using the above extended channel dependency graph construction, resource dependencies between multicast trees show up as cycles in the extended channel dependency graph even if they don't form cycles topologically. The cycles in the extended channel dependency graph can be broken by splitting a channel in the cycle into two virtual channels (or by adding another virtual channel if the physical channel has already been split). The added virtual channels are implemented in the corresponding routers. To decide on where to introduce virtual channels, we simply use a greedy heuristic of splitting the first channel encountered in each cycle. We readily admit that a more sophisticated optimization procedure could be envisioned for this virtual channel insertion problem. However, our simple greedy heuristic appears to suffice since we have found that virtual channels are rarely needed to resolve deadlocks in practice for custom networks.

Figure 7.1c shows that the extended channel dependency graph has a cycle between channel C and F. Thus we add a virtual channel C' to C and a virtual channel F' to F. By allowing flows coming from A to C and flows from E to C', flows from B go to F' and flows coming from D go to F, the deadlock is solved. Using this method, we find all cycles in extended CDG and add virtual channels as needed to guarantee deadlock free routing.

In all the benchmarks that we tested in our evaluations throughout this thesis, no deadlocks were found in any of the synthesized solutions. Therefore, we did not need to add any virtual channels to them. They were all verified to be deadlock-free.

Acknowledgement

This chapter is in part a reprint of the material in the paper: Shan Yan, Bill Lin, “Custom Networks-on-Chip Architectures with Multicast Routing”, *IEEE Transactions on VLSI Systems*, volume: 17, issue: 3, on pages: 342-355, March 2009. The dissertation author was the primary author of the above paper.

Chapter 8

Conclusion and Future Work

This thesis has demonstrated practical design methodologies and algorithms for the automated synthesis of custom Networks-on-Chip (NoC) architectures that are optimized to a given application. Besides data rate requirements, the proposed design methodologies and algorithms take into considerations user-defined objectives and constraints, implementation-specific design and process parameters, and floorplanning information. In addition, the proposed solutions are unique in comparison to prior work in that they consider both unicast and multicast flows, the latter being increasingly important to support applications such as cache coherence. Chapter 2 provided an overview to our proposed design flow and general problem formulation.

Specifically, two approaches to the NoC synthesis problem have been developed. The first approach is based flow-set partitioning and Steiner-tree construction, which was presented in Chapter 3. It is based on decomposing the problem into the inter-related steps of finding good flow-set partitions, deriving a good physical network topology for each group in the partition, and providing an optimized network implementation for the derived topologies. The second approach is based on a rip-up and re-route concept, which aims to produce increasingly better solutions through an iterative exploration process. At each iteration, a flow is ripped up from the current solution, with the network resources that it occupied released. Then this removed flow is re-routed over the remaining network based on formulating the multicast re-routing problem as a minimum directed spanning tree problem. This approach was described in Chapter 4. To our knowledge, our synthesis algorithms are among the first to optimize for multicast flows in the application-specific design domain.

The synthesis algorithms presented in these two chapters were evaluated on a variety of NoC benchmarks using power optimization as the primary objective. Experimental results showed that these algorithms can achieve 79.3% and 82.0% reduction in power consumption over different mesh implementations on unicast benchmarks and 47.9% and 50.5% reduction in power consumption on multicast benchmarks respectively. Significant improvements in performance were also achieved, with an average of 65.8% and 70.2% reduction in hop count on unicast benchmarks and 45.1% and 47.8% reduction in hop count on multicast benchmarks.

The proposed design methodologies and algorithms have also been extended to consider 3D chip designs and multiple use case applications. Specifically, while both NoC synthesis approaches presented in Chapters 3 and 4 produced high quality results, we found that the rip-up and re-route approach generally produced better results with less CPU times. Therefore, we chose to extend this approach to consider the additional design dimensions. The extensions for 3D NoC synthesis were presented in Chapter 5. The advantages of 3D silicon integration were explored in this chapter. Also, accurate power and delay models for 3D wiring with through-silicon vias were proposed along with efficient 3D-NoC synthesis algorithms that make use of these models. Experimental results showed that our 3D synthesis algorithms can on average achieve a 63% reduction in power consumption and a 17% in hop counts over different mesh implementations. To our knowledge, we are among the first to consider the 3D-NoC synthesis problems in the application-specific design domain.

The extensions to support multiple use case applications were presented in Chapter 6. Increasingly, multiprocessor SoCs are designed to support different usage scenarios since such SoC designs may be employed in different products or in products with different operation modes. For these embedded applications, the NoC must be designed to satisfy the communication characteristics and performance constraints of all usage scenarios considered, each with its own traffic profile. To support multiple traffic profiles, a concept of exclusive flow sets was introduced. Experimental results showed that our multi-profile synthesis algorithms can on average achieve a 35% reduction in power consumption and a 11% in hop counts over different mesh implementations. To our knowledge, we are among the first to consider multiple usage scenarios in the application-specific design domain.

Finally, Chapter 7 provided several mechanisms to ensure deadlock-free routing.

One mechanism is based on statically scheduled routing, and the other is based on virtual channel insertions.

8.1 Future Directions

Throughout this thesis, two assumptions have been made. The first is that the goal of NoC synthesis is to generate an efficient network architecture that can support the specified data rates. However, the proposed formulations do not directly address Quality-of-Service (QoS) issues such as real-time delay guarantees and bursty traffic. Accurate handling of these QoS issues will become increasingly important in future applications. Thus, extending our design methodologies and algorithms to synthesize NoCs with QoS support needs to be explored, with investigation into different QoS mechanisms.

The second assumption that needs to be relaxed is the assumption that the on-chip communication architecture should either be "bus-based" or "network-based". Although "buses" are inherently not scalable as a global communication fabric, they are still quite cost-effective for interconnecting small clusters of modules. They are simpler and cheaper to implement than full network solutions because they alleviate the need for routers. They also alleviate the need for "in-network" buffering since data is assumed to be held at the module until the module can acquired the bus. They are also inherently more power efficient for multicast operations if the communication remains local. One potentially interesting future direction to explore is the synthesis of "hybrid" multi-tier architectures that make use of local bus segments for "intra-cluster" communications and a global network that interconnects these bus segments for "inter-cluster" communications. Such hybrid multi-tier architectures can potentially lead to more power efficient and higher performance on-chip communication architectures.

Bibliography

[1] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. *In Proc. Design Automation and Test in Europe Conf.(DATE)*, pages 250-256, 2000.

[2] *The International Technology Roadmap for Semiconductors (ITRS)*, 2007.

[3] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and T. Todd. Surviving the SoC revolution. Kluwer Academic Publisher, 1999.

[4] ARM Inc. AMBA On-Chip Bus Standard. *http://www.arm.com/products/solutions/AMBAHomePage.html*.

[5] STBus from STMicrolectronics.

http://www.st.com/stonline/prodpres/dedicate/soc/cores/stbus.htm.

[6] *http://www.sonicsinc.com/*.

[7] IBM Inc. CoreConnect Bus Architecture. *http://www-3.ibm.com/chips/products/coreconnect*.

[8] D. Flynn. AMBA: enabling reusable on-chip designs. *IEEE Micro*, 17(4):20-27,July-Aug 1997.

[9] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. *In Proc. Design Automation Conf. (DAC)*, pages 684-689, June 2001.

[10] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on a chip: an architecture for billion transistor era. *In Proc. of the IEEE NorChip Conf.*, pages 166-173, Nov. 2000.

[11] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J. Soininen, M. Forsell, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. *In Proc. Symposium on VLSI*, pages 105-112, April 2002.

[12] Michael Keating and Pierre Bricaud. Reuse methodology manual for System-on-Chip designs. Kluwer Academic, 1998.

[13] W. Dally and J. Poulton. Digital systems engineering. Cambridge University Press, 1998.

[14] ARM Inc. ARM processor core overview. *http://www.arm.com/products/CPUs/.*

[15] M. B. Taylor et al., “The RAW microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 6, pp. 25-35, 2002.

[16] K. Sankaralingam et al. “Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," *ISCA*, 2003.

[17] L. Benini and G. De Micheli, “Networks on chips: A new SoC paradigm," *IEEE Computer*, pp. 70-78, January 2002.

[18] T. Lei and S. Kumar, “A two-step genetic algorithm for mapping task graphs to a network on chip architecture," *Proc. Euromicro Symp. Dig. Syst. Des.*, 2003, pp. 180-187.

[19] G. Ascia, V. Catania, M. Palesi, “Multi-objective mapping for mesh-based NoC architectures," *Proc. CODES/ISSS*, 2004, pp. 182-187.

[20] J. Hu, R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints," *ASP-DAC*, 2003.

[21] S. Murali and G. De Micheli, “Bandwidth constrained mapping of cores onto NoC architectures," *DATE,* 2004.

[22] U. Ogras, R. Marculescu, “Energy and performance driven NoC communication architecture synthesis using a decomposition approach," *DATE*, 2005.

[23] U. Ogras, R. Marculescu, “Application specific Network-on-Chip architecture customization via long range link insertion," *ICCAD*, 2005.

[24] A. Pinto, L. P. Carloni, A. L. Sangiovanni-Vincentelli, “Efficient synthesis of networks on chip," *ICCD,* 2003.

[25] K. Srinivasan, K. S. Chatha, G. Konjevod, “Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Transactions on VLSI Systems*, Volume 14, Issue 4 (April 2006), pp. 407-420.

[26] K. Srinivasan, K. S. Chatha, “ISIS: A genetic algorithm based technique for custom on-chip interconnection network synthesis", *International Conference on VLSI Design (IVLSI),* 2005.

[27] K. Srinivasan, K. S. Chatha, G. Konjevod, “Application specific Network-on-Chip design with guaranteed quality approximation algorithms," *ASPDAC 2007.*

[28] S. Murali, et al., “Designing application-specific networks on chips with floorplan information," *ICCAD,* 2006.

[29] D. Bertozzi, A. Jalabert, et al., “NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, Volume 16, Issue 2 (Feb 2005), pp. 113-129.

[30] A. S. Grove, “Changing vectors of Moore’s law," Keynote presentation, *International Electron Device Meeting*, December 2002.

[31] H. Wang et al., “Orion: A power-performance simulator for interconnection networks", *MICRO 35*, November 2002.

[32] X. Chen, L.-S. Peh, “Leakage power modeling and optimization in interconnection networks," *ISPLED*, 2003.

[33] G. Chen and E. G. Friedman, “Low-power repeaters driving RC and RLC interconnects with delay and bandwidth constraints," *IEEE Trans. on VLSI Systems*, Feb. 2006.

[34] L. Zhang, H. Chen, et al., “Repeated On-Chip Interconnect Analysis and Evaluation of Delay, Power, and Bandwidth Metrics under Different Design Goals", *ISQED 2007*

[35] L.-S. P. and W. J. Dally, “A delay model and speculative architecture for pipelined routers.”, *7th International Symposium on High-Performance Computer Architecture (HPCA)*, 2001

[36] H. Wang, L.-S. Peh and S. Malik, “Power-driven design of router microarchitectures in on-chip networks", *MICRO 36*, 2003.

[37] R. Mullins, “Minimising dynamic power consumption in on-chip networks," *International Symposium on System-on-Chip*, 2006.

[38] D. E. Knuth, The art of computer programming. Pre-Fascicle 3B. A draft of Sections 7.2.1.4-5: Generating all partitions. Addison-Wesley.

[39] D. M. Warme, P. Winter, M. Zachariasen, “Exact algorithms for plane Steiner Tree problems: A computational study," Advances in Steiner Trees, pp. 81-116, Kluwer Academic Publishers, 2000.

[40] http://www.diku.dk/geosteiner/

[41] C.-W. Lin, S.-Y. Chen, Ch.-F. Li, Y.-W. Chang, C.-L. Yang, “Efficient obstacle-avoiding rectilinear Steiner tree construction," *International Symposium on Physical Design*, 2007.

[42] N. A. Sherwani, Algorithms for VLSI Physical Design Automation, 3rd edition, Kluwer Academic Publishers, Norwell, MA, 1998.

[43] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.K. Cheng, and J. Gu, “Corner block list: An effective and efficient topological representation of non-slicing floorplan," *ICCAD*, 2000.

[44] S. N. Adya and I. L. Markov, “Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Transactions on VLSI Systems*, vol 11(6), pp. 1120-1135, December 2003.

[45] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, “Optimization by simulated annealing," *Science*, vol. 220, number 4598, pp. 671-680, 1983.

[46] D. Greenfield, A. Banerjee, et al., “Implications of Rent’s Rule for NoC Design and Its Fault-Tolerance," *NOCS 2007*, May 2007.

[47] D. Stroobandt, P. Verplaetse, J. van Campenhout, “Generating synthetic benchmark circuits for evaluating CAD tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Volume 19, Issue 9, Sep 2000 Page(s):1011 - 1022

[48] William A. Dees, Jr. and Patrick G. Karger “Automated rip-up and reroute techniques," *DAC*, 1982.

[49] Hyunchul Shin, Alberto L. Sangiovanni-Vincentelli, “A Detailed Router Based on Incremental Routing Modifications: Mighty," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 6, Issue 6, pp. 942-955, 1987

[50] H. Shirota, S. Shibatani, M. Terai, “A new rip-up and reroute algorithm for very large scale gate arrays," *ICICC*, May 1996

[51] Y. J. Chu and T. H. Liu, “On the shortest arborescence of a directed graph”, *Science Sinica*, v.14, 1965, pp.1396-1400.

[52] J. Edmonds, “Optimum branchings”, *Research of the National Bureau of Standards,* 71B, 1967, pp.233-240.

[53] S. Yan, B. Lin, “Application-Specific Network-on-Chip architecture synthesis based on set partitions and Steiner trees," *ASPDAC*, 2008

[54] E. Wein and J. Benkoski, “Hard macros will revolutionize SoC design," *EE Times*, August 20, 2004.

[55] “UMC Delivers Leading-edge 65nm FPGAs to Xilinx," *Design and Reuse*, November 8, 2006.

[56] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. W. Keckler, D. Burger, “Implementation and evaluation of a dynamically routed processor operand network," *NOCS 2007*, May 2007.

[57] S. Yan and B. Lin, “Design of application-specific 3D networks-on-chip architectures", *ICCD*, 2008.

[58] S. Yan, B. Lin, “Custom networks-on-chip architectures with multicast routing," *IEEE Transactions on VLSI Systems*, March 2009.

[59] S. Murali et al., “Mapping and configuration methods for multi-use-case networks on chips", *ASPDAC*, 2006.

[60] S. Murali, M. Coenen, et al., “A methodology for mapping multiple use-cases onto networks on chips", *DATE* 2006, pp. 118-123.

[61] Gang Cheng and Nirwan Ansari, “Finding a least hop(s) path subject to multiple additive constraints", *Computer Communications*, Feb 2006.

[62] D. Bailey, E. Barszcz, et al., “The NAS parallel benchmarks", *Technical Report NAS-95-020, NASA Ames Research Center*, 1995

[63] `http://www.intel.com/cd/software/products`

[64] Jose Duato, Sudhakar Yalamanchili, Lionel Ni, “Interconnection Networks," IEEE Computer Society, 1997

[65] B. Towles, W. J. Dally, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2003.

[66] W. J. Dally, C. L. Seitz, “Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, May 1987.

[67] J. Duato, “A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, 1995.

[68] X. Lin, P.K. McKinley, L.M. Ni, “Deadlock-free multicast wormhole routing in 2-D mesh multicomputers", *IEEE Transactions on Parallel and Distributed Systems*, Volume: 5, Issue: 8, Page(s): 793-804, Aug 1994

[69] M. P. Malumbres, J. Duato, J. Torrellas, “An efficient implementation of tree-based multicast routing for distributed shared-memory," *IEEE Symposium on Parallel and Distributed Processing*, 1996.

[70] Y. H. Song and T. M. Pinkston, “A progressive approach to handling message-dependent deadlock in parallel computer systems," *IEEE Transactions on Parallel and Distributed Systems,* vol. 14, no. 3, pp. 259ÍC275, 2003.

[71] Arteris, “A comparison of network-on-chip and busses," White paper, 2005.

[72] S. Murali and G. de Micheli, “An application-specific design methodology for STbus crossbar generation," In Proceedings of *Design, Automation and Test in Europe (DATE ąŕ05)*, vol. 2, pp. 1176ÍC1181, Munich, Germany, March 2005.

[73] SonicsMX Datasheet, Sonics, 2005, http://www.sonicsinc.com/.

[74] B. Gebremichael, F. Vaandrager, Z. Miaomiao, K. Goossens,E. Rijpkema, and A. Rćĺadulescu, “Deadlock prevention in the Æthereal protocol," in Proceedings of *the 13th IFIP WG 10.5 Advanced Research Working Conference Correct Hardware Design and Verification Methods (CHARME ąŕ05)*, pp. 345-348, Germany, October 2005.

[75] Z. Lu, B. Yin, and A. Jantsch, “Connection-orientedmulticasting in wormhole-switched networks on chip," in Proceedings of *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, pp. 205.210, Karlsruhe, Germany, March 2006.

[76] S. Murali, P. Meloni, F. Angiolini, et al., “Designing messagedependent deadlock free networks on chips for applicationspecific systems on chips," in Proceedings of *IFIP International Conference on Very Large Scale Integration*, pp. 158.163, Nice, France, October 2006.

[77] D. Starobinksi et al., “Application of network calculus to general topologies using turn-prohibition", *IEEE/ACM Transactions on Networking*, Vol. 11, Issue 3, pp. 411-421, June 2003.

[78] A. Hansson et al., “A Unified Approach to Mapping and Routing on a Combined Guaranteed Service and Best-Effort Network-on-Chip Architectures", Technical Report No: 2005/00340, Philips Research, April 2005.

[79] K. Goossens, J. Dielissen, and A. Rćadulescu, “The thereal network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, 2005.

[80] M. Millberg et al., “Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," *DATE*, 2004.

[81] F. A. Samman, T. Hollstein and M. Glesner, “Multicast parallel pipeline router architecture for network-on-chip", *DATE 2008*, 2008.

[82] E.A. Carara, F.G. Moraes, “Deadlock-Free Multicast Routing Algorithm for Wormhole-Switched Mesh Networks-on-Chip," *ISVLSI*, 2008.

[83] K. Lee et al. “Three-Dimensional Shared Memory Fabricated using Wafer Stacking Technology," *IEDM Technical Digest*, Dec. 2000.

[84] L. Xue, C. C. Liu et al., “Three Dimensional Integration: Technology, Use, and Issues for Mixed-Signal Applications," *IEEE Trans. on Electron Devices,* 50:601-609, May 2003.

[85] W. R. Davis et al. “Demystifying 3D ICs: The Pros and Cons of Going Vertical," *IEEE Design & Test of Computers*, 22(6):498-510, 2005.

[86] M. Kawano, S. Uchiyama et al., “A 3D Packaging Technology for 4Gbit Stacked DRAM with 3Gbps Data Transfer," *IEEE Int. Electron Devices*, pp. 1-4, 2006.

[87] B. Black, D. Nelson et al., 3D Processing Technology and Its Impact on IA32 Microprocessors. *ICCD*, 2004.

[88] T. Kgil et al. PICOSERVER: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor. ASPLOS-XII, 2006.

[89] F. Li, C. Nicopoulos et al., Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In 33rd International Symposium on Computer Architecture (ISCA), pages 130-141, 2006.

[90] K. Bernstein, P. Andry et al., Interconnects in the Third Dimension: Design Challenges for 3D ICs, *DAC*, 2007.

[91] P. Morrow, B. Black et al., Design and Fabrication of 3D Microprocessor, Material Research Soc. Symp. 2007.

[92] J. Cong, J. Wei, and Y. Zhang, “Thermal-Driven Floorplanning Algorithm for 3D ICs," *ICCAD*, 2004.

[93] B. Goplen and S. Sapatnekar, “Efficient Thermal Placement of Standard Cells in 3D ICs using a Force Directed Approach," *ICCAD*, 2003.

[94] W.-L. Hung, G.M. Link, et al., “Interconnect and thermal-aware floorplanning for 3D microprocessors," *ISQED*, 2006.

[95] C. Addo-Quaye, Thermal-aware mapping and placement for 3-D NoC designs, IEEE International SOC Conference, 2005.

[96] J. Cong, and Y. Zhang, “Thermal-Driven Multilevel Routing for 3-D ICs," *ASPDAC*, 2005.

[97] Mohit Pathak, Sung Kyu Lim, “Thermal-aware Steiner Routing for 3D Stacked ICs," *ICCAD*, 2007.

[98] V. F. Pavlidis, E. G. Friedman, 3-D Topologies for Networks-on-Chip, *IEEE Transactions on VLSI Systems*, Oct. 2007.

[99] H. Matsutani, M. Koibuchi, H. Amano, Tightly-Coupled Multi-Layer Topologies for 3-D NoCs, International Conference on Parallel Processing (ICPP), 2007.

[100] J. Kim, C. Nicopoulos et al., A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures, *Proceedings of the International Symposium on Computer Architecture*, 2007.

[101] E. Rijpkema et al., “Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," *DATE*, 2003.

[102] N. Enright-Jerger, M. Lipasti and L.-S. Peh, “Circuit-switched coherence", *IEEE Computer Architecture Letters*, vol. 6, no. 1, Mar. 2007.