UNIVERSITY OF CALIFORNIA, SAN DIEGO

Interconnection Networks Synthesis and Optimization

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

by

Yi Zhu

2008

.

The dissertation of Yi Zhu is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2008

To My Parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Professor Chung-Kuan Cheng, for his constant support and help. Without his help and encouragement, I could not go through all the difficulties and have such a proud achievement in my life. Besides his help, I am also impressed by his confidence, persistence, diligence, passion, and smartness. It has been a great honor to have the opportunity to work with him and learn from him.

I wish to thank my dissertation committee members, Professor Fan Chung Graham, Professor Bill Lin, Professor Michael Taylor and Professor Steven Swanson for technical discussions and their advices and reviews of this dissertation.

I am grateful to all the graduate students in the UCSD VLSI CAD group for making the group a friendly and fun place to work. Among them, special thanks to Bo Yao, Hongyu Chen, Zhengyong Zhu, Shuo Zhou, Jianhua Liu, Yuanfang Hu, Haikun Zhu, Rui Shi, He Peng, Ling Zhang, Wanping Zhang, Renshen Wang, Amirali Shayan, Yulei Zhang, Xiang Hu and many others.

In addition, I would like to thank Kian Win Ong and Thomas Weng for their valuable suggestions on this work.

Finally my special thanks go to my parents, for their support during my education and for their understanding and tolerance during the last couple of years.

Chapter III includes the contents of two published papers."Communication Latency Aware Low Power NoC Synthesis," by Y. Hu, Y. Zhu, H. Chen, R. Graham, C.K. Cheng, in Proceedings of 43rd ACM/IEEE Design Automation Conference. "Physical Synthesis of Energy-Efficient NoCs Through Topology Exploration and Wire Style Optimization," by Y. Hu, H. Chen, Y. Zhu, A. A. Chien, C.K. Cheng, in Proceedings of 23th IEEE International Conference of Computer Design. The dissertation author was the researcher and co-author of both papers.

Chapter IV includes the contents of one paper "Advancing Supercomputer Performance Through Interconnection Topology Synthesis", by Y. Zhu, M.

Taylor, S. B. Baden, C.K. Cheng, to appear in Proceedings of International Conference on Computer Aided Design, 2008. The dissertation author was the primary researcher and co-author of the paper.

VITA

2003	B.Comp. (1st class honors) in Computer Science National University of Singapore, Singapore
2006	M.S. in Computer Science University of California, San Diego
2008	Ph.D. in Computer Science University of California, San Diego

PUBLICATIONS

Yi Zhu, Amirali Shayan, Wanping Zhang, Tzyy-Ping Jung, Jeng-Ren Duann, Scott Makeig and Chung-Kuan Cheng, "Analyzing High-Density ECG Signals Using ICA", *IEEE Transactions on Bio Medical Engineering (TBME)*, accepted

Yi Zhu, Thomas Weng and Chung-Kuan Cheng, "Enhancing Learning Effectiveness in Digital Design Courses Through the Use of Programmable Logic Boards", *IEEE Transactions on Education (TE)*, accepted

Yi Zhu, Yuanfang Hu and Chung-Kuan Cheng, "Energy and Switch Area Optimizations for FPGA Global Routing Architectures", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, accepted

Yi Zhu, Michael Taylor, Scott B. Baden and Chung-Kuan Cheng, "Advancing Supercomputer Performance Through Interconnection Topology Synthesis", *International Conference on Computer Aided Design (ICCAD 2008)*

Rui Shi, Wenjian Yu, Yi Zhu, Chung-Kuan Cheng and Ernest S. Kuh, "Efficient and Accurate Eye Diagram Prediction for High Speed Signaling", *International Conference on Computer Aided Design (ICCAD 2008)*

Yi Zhu, Jianhua Liu, Haikun Zhu and Chung-Kuan Cheng, "Timing-Power Optimization for Mixed-Radix Ling Adders by Integer Linear Programming", *13th Asia and South Pacific Design Automation Conference (ASPDAC 2008)*

Wanping Zhang, Yi Zhu, *et al.*, "Finding the Worst Case of Voltage Violation in Multi-Domain Clock Gated Power Network with an Optimization Method", *7th Design, Automation and Test in Europe (DATE 2008)*

Renshen Wang, Evangeline F. Y. Young, Yi Zhu, Fan Chung Graham, Ronald Graham and Chung-Kuan Cheng, "3-D Floorplanning Using Labeled Tree and Dual Sequences", *International Symposium on Physical Design (ISPD 2008)*

Jianhua Liu, Yi Zhu, Haikun Zhu, John Lillis and Chung-Kuan Cheng, “Optimum Prefix Adders in the Space of Timing, Power and Area”, *12th Asia and South Pacific Design Automation Conference (ASPDAC 2007)*

Shuo Zhou, Bo Yao, Hongyu Chen, Yi Zhu, *et al.*, “Efficient Timing Analysis with Known False Paths Using Biclique Covering”, *IEEE Transactions on CAD (TCAD)*, 26(5), 959–969, 2007

Yuanfang Hu, Yi Zhu, Hongyu Chen, Ronald Graham and Chung-Kuan Cheng, “Physical Synthesis of Communication Latency Aware Low Power NoCs”, *43rd Design Automation Conference (DAC 2006)*

Yuanfang Hu, Yi Zhu, Michael Taylor and Chung-Kuan Cheng,“FPGA Global Routing Architecture Optimization Using a Multicommodity Flow Approach”, *25th International Conference of Computer Design (ICCD 2007)*

Haikun Zhu, Yi Zhu, David Harris and Chung-Kuan Cheng, “An Interconnect-Centric Approach to Cyclic Shifter Design Using Fanout Splitting and Cell Order Optimization”, *12th Asia and South Pacific Design Automation Conference (ASPDAC 2007)*

Shuo Zhou, Yi Zhu, Yuanfang Hu, Ronald L. Graham, Mike Hutton and Chung-Kuan Cheng, “Timing Model Reduction for Hierarchical Timing Analysis”, *International Conference on Computer Aided Design (ICCAD 2006)*

Yi Zhu, Tong Lee Chen, Wanping Zhang, Tzyy-Ping Jung, Jeng-Ren Duann, Scott Makeig and Chung-Kuan Cheng, “”Noninvasive Study of the Human Heart using Independent Component Analysis”, *6th IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2006)*

Yuanfang Hu, Hongyu Chen, Yi Zhu, Andrew A. Chien and Chung-Kuan Cheng, “Physical Synthesis of Energy-Efficient Networks-on-Chip Through Topology Exploration and Wire Style Optimization”, *23th International Conference of Computer Design (ICCD 2005)*

Shuo Zhou, Bo Yao, Hongyu Chen, Yi Zhu, *et al.*, “Improving the Efficiency of Static Timing Analysis with False Paths”, *International Conference on Computer Aided Design (ICCAD 2005)*

FIELDS OF STUDY

Major Field: Computer Science
Studies in VLSI CAD
Professor Chung-Kuan Cheng

ABSTRACT OF THE DISSERTATION

Interconnection Networks Synthesis and Optimization

by

Yi Zhu
Doctor of Philosophy in Computer Science
University of California, San Diego, 2008
Professor Chung-Kuan Cheng, Chair

The advent of new technologies brings revolutions in the fields of VLSI design and high performance computing. On one hand, the increasing number of processing elements, both in on-chip multi-core systems and supercomputer systems, demands high bandwidth communications. On the other hand, the performance of the system, usually measured by the latency and power consumption, is gradually being dominated by the interconnection networks. These facts raise challenges in synthesizing and optimizing interconnection networks.

In this dissertation, we study methodologies and algorithms to perform the interconnection network synthesis and optimization in both on-chip networks and supercomputer systems. We explore a wide range of network topologies and physical implementations, and evaluate the performance of multi-commodity flow (MCF) algorithms. We design efficient approximation schemes to solve different variations of MCF problems, which incorporate different practical constraints. The automated design flows discover much larger design space than the traditional methods and therefore achieve promising results.

In the study of Network-on-Chip (NoC), we are optimizing the communication latency and power consumption, which are two competing design objectives. With an improved fully polynomial approximation algorithm, power optimal design of a structured 8×8 NoC can be found for given average latency constraints with certain communication bandwidth requirements. Our methodology explores a large number of topologies, introduces a variety of wire styles into NoC design, and incorporates latency constraints and power minimization objectives into a unified MCF model, with simultaneous optimization on network topologies, physical embedding, and interconnect wire styles. The results demonstrate the strengths of the optimized networks and indicate the clear trend of power and latency tradeoffs.

In the synthesis and optimization of networks in supercomputer systems, we use the packaging framework of the Blue Gene/L supercomputer as an example to demonstrate the advantages of our design flow, which has incorporated real design issues, such as board dimensions and pin numbers. Using real benchmark traces, the experiments show that the best topologies identified by our algorithm can achieve better average latency compared to the existing 3-dimensional torus networks.

I

Introduction

Interconnection networks have become the core component of modern digital systems. According to [14], a digital system consists of three basic building blocks: logic, memory and communication. Logic transforms and combines data, memory stores data for later retrieval, and communication moves data from one location to another. The performance of most digital systems today is limited by their communication or interconnection. That is because as technology advances, according to Moore's Law, processors and memory are becoming smaller, faster and cheaper. The speed of light, however, remains unchanged. Therefore, most of the clock cycles are spent on wire delay instead of gate delay, and most of the power is used to drive the wires. Therefore, in order to improve the system performance, designers have to make efficient use of scarce interconnection resources, such as pin density and chip/board dimensions. Thus the study of interconnection networks has been considerably important.

We notice the following two categories of interconnection networks among processors and other system components, which have been extensively studied in literature: the networks for on-chip interconnection architectures, and the networks in computer clusters. The first category is usually considered as Network-on-Chip. The applications in the second category are usually the modern supercomputers, e.g. IBM Blue Gene/L, or Cray BlackWidow. Although the design objectives are usually similar between these two categories, e.g. high performance and low power, different design issues should be addressed. For example, the on-chip interconnection networks usually have fewer processing elements and wires, and constrained routing areas. The networks in a supercomputer, on the other hand, contain a large number of processors, and are more constrained by the board pins. There have been a lot of case studies on the interconnection networks in these two categories.

In this dissertation, we propose a generic algorithmic approach to synthesize and optimize the interconnection networks, both for on-chip architectures and multi-processor systems. This approach formulates the problem using a multi-commodity flow (MCF) model and evaluates the networks with different topologies using efficient algorithms. Different design issues are reflected as constraints in the formulation. This methodology enables us to explore a wide variety of topologies instead of several regular ones. Network topology is a crucial factor that affect the performance of interconnection networks. We show two case studies in this dissertation, which involve NoC and supercomputer respectively.

Section I.1 will explain the motivation of this work. Section I.2 to Section I.4 will review the previous works in different areas, including the on-chip interconnection networks, the supercomputer interconnection networks, and the interconnection network topologies. Section I.5 will sketch the organization of the dissertation.

I.1 Motivation

In the past decades, Moore's Law continues to be valid and semiconductor technology keeps scaling down. This allows processors and memories to become faster and smaller. On the other hand, as the speed of light remains unchanged, and wiring density is scaling at a slower rate than the components, the interconnection in most digital systems have become the bottleneck [14]. Prior to the early 1990s, delay and power consumption of logic gates dominated, and on-chip wires were considered as purely capacitive loads of logic gates in chip design. Nowadays, growing wire resistance coupled with shrinking native gate speed have made wire delays and power consumption increasingly important.

Thus, a good design of interconnection networks is essential to guarantee the performance in most digital systems. In literature, there have been a lot of case studies on different interconnection networks, in both on-chip systems, and clustered supercomputers (see Section I.2 and Section I.3 respectively). However, designers are still facing quite a few challenges:

1. The design space of interconnection networks is huge. The network topology, as one of the most important factors in interconnection network synthesis, has a large number of choices (see Section I.4). Thus, it is a challenging question for designers to make a good choice. The wire style, which is another design choice, is also critical: different wires, such as RC wires and transmission lines, possess different properties in terms of delay, power and width, and therefore have a large impact on the system performance. To take all the factors into consideration will be a tough challenge for system designers.

2. The limiting resources for interconnection networks add difficulties in designing efficient networks. Based on Moore's Law, the area of components become smaller while the number of elements keep increasing. Thus, an increasing number of wires need to be planned on a decreasing routing area. The same issue exists for processors among boards and clusters. the pins, as the interfaces of the boards, are scarce resources. Thus, careful planning of wires is necessary.

3. There is no general framework to synthesize and optimize interconnection networks with different parameters, such as the scale of networks, the physical constraints, and the communication patterns. Although there are a lot of successful case studies, most of them could not be transplanted to other applications, which prevents their methodologies from being widely adopted.

In this work we will propose a general methodology which makes use of algorithmic approaches to address the above issues.

I.2 Review of On-Chip Interconnection Networks

Network-on-Chip (NoC) has been proposed [49] [13] [36] as an attractive alternative to traditional dedicated wires to achieve high performance and modularity. With the advance of semiconductor technology, we have observed that more IP blocks, such as processors, memory subsystems and DSPs, are integrated on a single chip and interconnected by NoCs [26].

Power efficiency and communication latency are two main concerns in NoC design. On one hand, we hope to deliver packets to their destinations within the shortest possible period. Adding long shortcut links to regular mesh based topology is an effective approach to achieve this by reducing the number of traffic hops in NoC [42]. On the other hand, power consumption has become one of the first order design considerations of the nano-scale VLSI designs. Extra shortcut links will lead to more complicated router architecture, which can potentially worsen the NoC power consumption. Our work studies the tradeoffs between NoC power efficiency and its average communication latency.

We focus on two design choices. NoC topology selection and interconnect wire style optimization, to optimize NoC power consumption and latency simultaneously. At the same time, the design satisfies given constraints, such as communication bandwidth requirements and physical on-chip area resource constraints.

The choice of network topology is important in NoC design. Different NoC topologies can dramatically affect the network characteristics, such as number of hops, total wire length, and communication flow distributions. These characteristics in turn determine the latency and power efficiency of NoC architectures.

Wire style optimization also plays an important role. Optimized wire styles for the critical NoC links can significantly reduce NoC power consumption and latency. Recent advances in signaling interconnect technologies, such as wave-pipelined RC wires with repeated buffers, low-swing differential pairs, and on-chip transmission lines, provide us with wiring schemes to optimize aggressively. These technologies along with traditional minimal separated RC wires display different tradeoffs between wire resources, wire delay and power consumption.

In recent years, researchers have studied NoC design issues to improve latency and power consumption. In [26], Hu et *al.* proposed a branch and bound algorithm to map the processing cores onto a tile-based mesh NoC architecture to satisfy bandwidth constraints and minimize total energy consumption. In [41] [40], Murali et *al.* designed topology mapping to minimize the average communication delay while satisfying bandwidth constraints, and presented a tool named SUNMAP to select the best topology for a given application. In [54], a variety of NoC topologies are designed and the effect of topology on NoC power consumption is studied. In [42], Ogras et *al.* inserted a few application-specific long-range links to regular mesh based topology to reduce average packet latency.

I.3 Review of Interconnection Networks in Supercomputers

Interconnection networks play an important role in the multiprocessor system. Currently, massively parallel computer systems have become popular, where the interconnection networks consist of a lot of processing cores. For example, IBM Blue Gene/L has 65,536 processors located in 64 racks [16] and the Cray Black Widow networks scales up to 32K processors [47]. Thus, the interconnection networks in these systems have become more of a critical factor in the performance of a computer system than the computing or memory modules [12]. Communication latency, which largely depends on the interconnection network, is of great concern in these current multiprocessor systems and has become crucial with the growth of system sizes and the shrinking of clock cycles [47].

Most interconnection networks in the current multiprocessor systems make use of regular low-radix topologies, among which the k-ary n-cube [11] and torus topologies are the most often used [47]. The examples includes the SGI Origin2000 hypercube [37], the Cray X1's dual-bristled, sliced 2D torus [3], the Cray T3E and XT3's 3D torus [48] [4], the Alpha 21364's torus networks [39], and recently, the Blue Gene/L's 3D torus networks [18]. In the recent Cray BlackWidow system, the network uses a high-radix folded Clos and fat-tree topology with side links [47].

In all the aforementioned systems, the topologies of their interconnection networks are limited to several well-known regular structures and determined by the designers. Although the simpler topologies imply simpler design and manufacture, they are probably unable to capture the bottleneck of the communications among processors and fully utilize the limited interconnect resources (e.g. wires, pins, connectors), therefore affecting the performance of the entire system. For example, if similar communication patterns often occur, a special designed topology is always preferred. For instance, topologies with long links are preferred when there are a lot of communications among processors located far away from each other, since the number of hops among these processors can be reduced. In particular, because high-radix routers appear more often these days [34], we have more flexibility to design the interconnection topologies to enhance the performance.

I.4 Review of Network Topologies

Network topology refers to the static arrangement of channels and nodes in an interconnection network [14]. The choices of topologies are of importance for interconnection network synthesis, since the topology is not only related to the type of the network, but also the details, such as the radix of routers and the bandwidth. Thus, we have to choose a topology based on the cost and performance.

Various kinds of topologies have been studied and applied in literature.

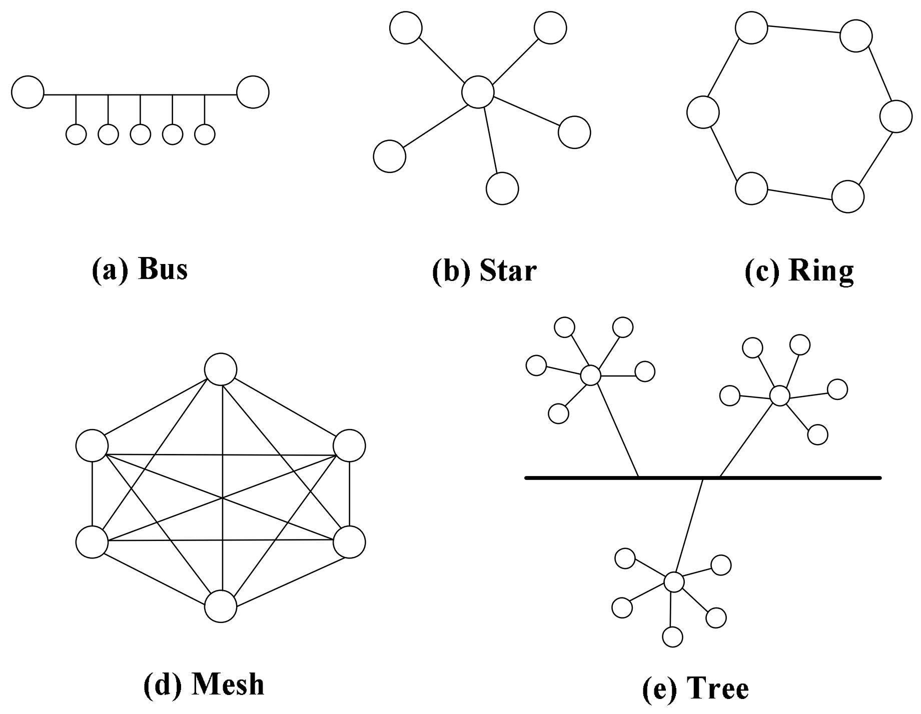


Figure I.1: Five Classic Topologies

Generally, there are the following several classic topologies:

1. *Bus*: A bus network topology is a network architecture in which a set of clients are connected via a shared communications line (Figure I.1(a)). Bus networks are the simplest way to connect multiple clients, but often have problems when two clients want to transmit at the same time on the same bus. Buses are easy to implement and extend, and consume less cable and therefore is cheaper than other topologies. Taking the above features into consideration, buses are usually suitable for temporary or small networks which do not require high speeds.

2. *Star*: A star network is composed of one central switch, hub or computer, which acts as a conduit to transmit messages (Figure I.1(b)). The star topology reduces the chance of network failure by connecting all of the systems to a central node. Compared with buses, stars have better performance, good isolation of each device, and are easy to scale by adjusting the capacity of

the central hub. On the other hand, the biggest disadvantage of stars is the high dependence of the central hub.

3. *Ring*: A ring network is a network topology in which each node connects to exactly two other nodes, forming a circular pathway for signal (Figure I.1(c)). It is a very orderly network where every device has access to the token and the opportunity to transmit. It also performs better than a star topology under heavy network load, and no network server is required. However, one malfunctioning workstation or bad port can create problems for the entire network. In addition, moving, adding, or changing of devices can affect the network.

4. *Mesh*: A mesh network is a type of network setup where each of the computers and network devices are interconnected with one another, allowing for most transmissions to be distributed, even if one of the connections go down. In a full mesh network, nodes are all connected to each other is a fully connected network (Figure I.1(d)). In a partial mesh network, some of the nodes are connected to more than one other node in the network with a point-to-point link — this makes it possible to take advantage of some of the redundancy that is provided by a physical fully connected mesh topology without the expense and complexity required for a connection between every node in the network.

5. *Tree*: A Tree Network consists of star-configured nodes connected to switches/concentrators, each connected to a linear bus backbone (Figure I.1(e)). In a tree network, we are able to use point-to-point wiring for individual segments. It is supported by several hardware and software venders. However, the overall length of each segment is limited by the type of cabling used. The robustness is also an issue in tree networks: if the backbone line breaks, the entire segment goes down.

Among these topologies, the mesh topology is widely used in traditional networks, because of the simple connection and easy routing provided by adjacency [11]. Furthermore, as high-radix routers are more and more desired [12], topologies with high node degrees, such as k-ary n-dimensional mesh or k-ary n-dimensional torus, are well studied [11].

There are also other types of network topologies: De Bruijn graph [15], Kautz graph, Circulant graph, butterfly graph, pyramid graph, and clos network [10]. All of them are regular topologies that could be generated in systematic ways. Recently, in both on-chip architectures and supercomputers, new variations of classic topologies were developed, for example, a flattened butterfly in NoC [33], and a high-radix folded Clos and fat-tree topology with side links in the BlackWidow supercomputer [47]. They possess different properties and have different advantages/disadvantages.

In literature, these topologies were all proposed and applied individually in different case studies, i.e. no one has ever made a comprehensive comparison using real applications. In this work, we are able to evaluate a large number of different network topologies, besides the ones mentioned above. The topology generation scheme will be described in detail in Section III.3.B.

I.5 Dissertation Organization

In this dissertation, we describe an algorithmic methodology that performs the interconnection network synthesis and optimization among processors in chips as well as among clusters. This work makes contributions both in algorithm theory and in system designs.

Chapter II discusses the theoretical issues of this work. The MCF algorithms, which form the core of our methodology, are depicted in detail. We first give the formulation of the MCF problem and then review the prevailing algorithms in literature. Then we introduce our polynomial approximation schemes, which take the practical constraints into consideration. We also propose an interval estimation heuristic to improve the performance, without compromising the accuracy of the algorithms.

Chapter III and IV study the applications of our approach in on-chip networks and supercomputers respectively. In Chapter III, the objective is to find the most power efficient NoC topologies and their wire style assignments, where the latency and bandwidth requirements are met. With efficient MCF algorithms, power optimal design of a structured 8×8 NoC can be found for given average latency/pairwise constraints with certain communication bandwidth requirements. Our methodology not only introduces a variety of wire styles into NoC design, and incorporates latency constraints and power minimization objectives into a unified MCF model, but also explores a large design space of network topologies.

In Chapter IV, we show a design flow to discover the best topology in terms of communication latency and physical constraints. First a set of representative candidate topologies are generated for the interconnection networks among computing chips. Afterwards an efficient multi-commodity flow algorithm is devised to evaluate the performance. The experiments show that the best topologies identified by our algorithm can achieve better average latency compared to the existing networks.

Finally, Chapter V summarizes this dissertation, discusses the strengths and weaknesses of our methodologies, and sketches some promising research directions.

II

Multi-commodity Flow Algorithms

In this section, we will introduce the algorithms for multi-commodity flow (MCF) problems, which are the core of our solvers in interconnection synthesis and optimization. Section II.1 will introduce the basic formulation. Section II.2 will discuss the previous efforts on this problem. In Section II.3 we shall introduce the algorithms we develop to tackle the synthesis and optimization. Finally, Section II.4 will discuss the future work in this theoretical direction.

II.1 Introduction

Multi-commodity flow (MCF) problems are considered as fundamental graph problems and have received much attention from researchers since the 1960s. The pioneers were Ford and Fulkerson [21], and Hu [27]. After that, a lot of algorithms were proposed to solve MCF either optimally or approximately. Here, the word MCF only denotes a category of problems, and various notations and formulations are used in the literature. To present the problems and algorithms clearly, we first unify the basic notations as follows:

- The given directed graph network is $G(V, E)$, where the number of nodes is

$|V| = n$, and the number of edges is $|E| = m$;

- Each edge in G has a capacity $u(e)$ and a cost $c(e)$. $u(e)$ and $c(e)$ are also denoted as u_{ij} and c_{ij} if $e = (i, j)$;
- There are K commodities, each has a source s_k and a sink t_k, and a demand $d(k)$.

If different constraints and objectives are imposed, different versions of MCF can be obtained. The following problems are the most often used:

- *maximum multi-commodity flow problem*: when there are no commodity demands and edge costs, and the total flow of the K commodities needs to be maximized;
- *maximum concurrent flow problem*: each commodity k is routed $\lambda d(k)$ units under the capacity constraints, the throughput λ needs to be maximized;
- *min-cost multi-commodity flow problem*: $d(k)$ units are needed to be routed for each commodity k under the capacity constraints, and the objective is to minimize the total cost;

There are also variants in addition to the above. For instance, a simpler version of the maximum concurrent flow problem has the same capacities for all the edges, which is called *uniform capacity maximum concurrent flow problem.* And in many research papers, the min-cost multi-commodity flow problem is formulated as the maximum concurrent flow problem with a cost budget constraint. We will discuss these in details in the later sections.

Figure II.1 shows an example of the maximum concurrent flow: assume there are two commodities, one flow from $s1$ to $t1$, the other flow from $s2$ to $t2$, each has the demand 1; the edge capacities are shown in the edges. The optimal throughput should be 1.5 — there are 1.5 multiples of commodity 1 and 1.5 multiples of commodity 2 routed and the middle edge with capacity 3 saturated.

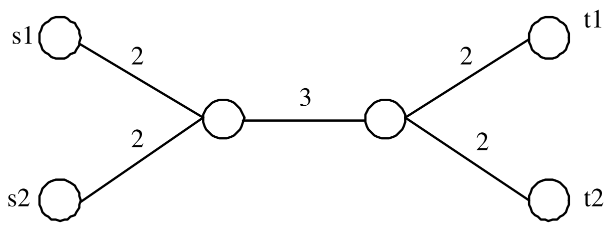


Figure II.1: A Maximum Concurrent Flow Example

All the aforementioned variants can be solved in polynomial time. A simple observation is that all of them can be formulated as linear programs. For example, let x_{ij}^k denote the flow of commodity k in the edge (i, j), the *maximum concurrent flow* can be formulated as follows:

Maximize

$$\lambda$$

Subject to

$$\sum_{1 \le k \le K} x_{ij}^k \le u_{ij} \tag{II.1}$$

$$\sum_{j:(i,j) \in E} x_{ij}^k - \sum_{j:(j,i) \in E} x_{ji}^k \ge \lambda d(k), \forall i = s_k \tag{II.2}$$

$$\sum_{j:(i,j) \in E} x_{ij}^k - \sum_{j:(i,j) \in E} x_{ji}^k = 0, \forall i \in V - \{s_k, t_k\} \tag{II.3}$$

$$\sum_{j:(i,j) \in E} x_{ij}^k - \sum_{j:(i,j) \in E} x_{ji}^k \ge -\lambda d(k), \forall i = t_k \tag{II.4}$$

The other versions can be formulated in a similar manner. We know that optimal solutions of linear programs can be found in polynomial time, so are these variants. However, the practical complexities of the linear programming (LP) algorithms prevent themselves from being used in practice when the problem scales become moderate or large — the best LP methods to solve MCF is the interior point method due to Vaidya [52] and has the complexity $O(K^{2.5}n^2m^{0.5}\log(nB))$, where B is the largest demand/capacity value. This is much slower than the algorithms we will introduce later.

II.2 Previous Works

In this section, we shall discuss the existing approximation schemes for MCF problems. As mentioned in the introduction part, these algorithms are able to achieve a certain error bound — for instance, an approximation scheme for the maximum concurrent flow problem is able to find a solution which is no smaller than $(1 - \epsilon)$ of the real optimal solution. Therefore, any proposed approximation algorithm has to be shown that the solution it produces can reach such a bound in a certain number of iterations (bounding the running time can usually be achieved by bounding the number of iterations). So based on the methods used to prove the bounds, the existing available approximation algorithms can be divided into two categories: the first category of algorithms make use of the primal-dual theory in LP and the second category of algorithms borrow the concepts of "potential" from physics to complete the proof. The primal-dual algorithms were developed well in recent years and become prevailing. Thus we would only introduce them here.

The first category of algorithms were originated from the early 1990s and constitute the mainstream of current MCF algorithms, due to its accuracy and efficiency. The core idea of these algorithms are from the primal-dual theory in LP: for a maximization problem like maximum concurrent flow, a primal feasible solution λ is always no greater than the optimal solution OPT, and a dual feasible solution D is always no less than the optimal solution OPT — so we have $\lambda \leq OPT \leq D$. The algorithms start from a feasible primal solution a feasible dual solution, update iteratively — increase the primal solution and decrease the dual solution — till the gap between the primal and dual solutions are within a certain small bound. Figure II.2 shows the basic idea. Note that in a minimization problem, the positions of primal and dual values will be reversed. ϵ is usually called the duality gap and uses relative measure $(D - \lambda)/D$. Furthermore, in the process we do not require the primal (dual) values be monotonically increasing (decreasing) with the iterations, as shown in the figure. In fact, oscillations do happen in many circumstances. We just need to ensure the gap ϵ is small enough

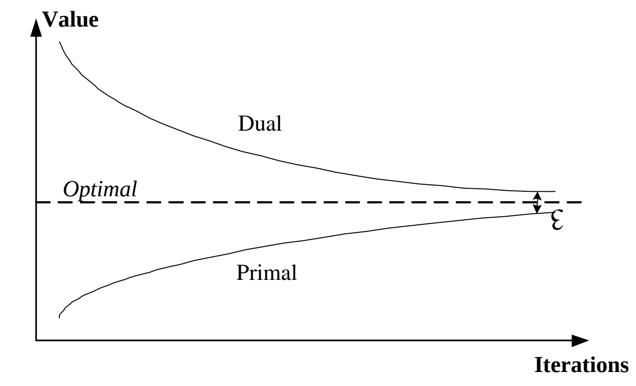


Figure II.2: The Basic Idea of the Primal-Dual Based Algorithms

when algorithms terminate.

To illustrate the concept more clearly, let's give the primal and dual formulations of the maximum concurrent flow problem. In these formulations, the path-based formulation is always adopted instead of the edge-based formulation presented in section 1. The decision variable $x(P)$ represents the flow amount in path P, and the notation P_k represents the set of paths for commodity k. The primal formulation is shown as follows:

Maximize

$$\lambda$$

Subject to

$$\sum_{P:e\in P} x(P) \leq u(e), \forall e \tag{II.5}$$

$$\sum_{P\in P_k} x(P) \geq \lambda d(k), \forall k \tag{II.6}$$

The corresponding dual formulation can be written as follows, where the dual variables $l(e)$ correspond to constraint (II.5), and z_k correspond to constraint (II.6).

Minimize

$$\sum_{e \in E} u(e)l(e)$$

Subject to

$$\sum_{e \in P} l(e) \geq z_k, \forall k, \forall P \in P_k \tag{II.7}$$

$$\sum_{k=1}^{K} d(k) z_k \geq 1 \tag{II.8}$$

The dual variables $l(e)$ can be treated as the lengths of the edges. In each iteration, the flows on the paths $X(P)$ and the edge lengths $l(e)$ are updated accordingly so that the primal and dual solutions are guaranteed to converge to a small gap in limited number of iterations.

The algorithms in this category can be further divided into two types according to the two milestones: Shahrokhi and Matula [50] first proposed this idea and reroute flows in each iteration to achieve the objective; Garg and Konemann [22] made the breakthrough by augmenting flows instead of rerouting them therefore ended up with a much simpler algorithm. In the following we shall discuss these two types of works respectively.

II.2.A Flow Rerouting Algorithms

The first paper that starts the research of combinatorial polynomial time approximation algorithms was by Shahrokhi and Matula [50]. This paper has two major contributions: first, it sets up the duality theory of the maximum concurrent flow problem; second, it provides a rerouting flow approximation algorithm to solve the uniform capacity maximum concurrent flow problem.

Based on the primal-dual theory, Shahrokhi and Matula derived the following three lemmas, assuming f is a maximum concurrent flow and l is an optimal distance function:

1. *all active paths for f are shortest paths under l*;

2. *edges assigned nonzero distance by l are saturated by f;*

3. *any optimal distance function l assigns nonzero distances only on critical edges.*

To complete the duality theory for the maximum concurrent flow problem, the authors further introduce two definitions:

- Let $G = (V, E)$ be a graph; denote by $(A_1, A_2, \ldots, A_k)$ for $k \geq 2$, the set of all edges in G whose end vertices are in distinct elements of the partition $\{A_1, A_2, \ldots, A_k\}$ of V, then $(A_1, A_2, \ldots, A_k)$ is called a *k-partite cut* in G.

- The *density* of the k-partite cut $(A_1, A_2, \ldots, A_k)$ is defined by
$$den(A_1, A_2, \ldots, A_k) = \min_l \frac{\sum_{1\leq i<j\leq k} l_{ij} U(A_i, A_j)}{\sum_{1\leq i<j\leq k} l_{ij} D(A_i, A_j)}$$

Then they prove the following important theorem called Max-concurrent flow Min k-partite cut Theorem: *For any maximum concurrent flow problem, the maximum throughput of any concurrent flow equals the minimum density of any k-partite cut, and in particular, equals the density of the k-partite cut of critical edges.* It can be proven by the previous three lemmas.

Based on the above theory, the comparison is made between the single commodity flow and the concurrent flow problems, as summarized in Table II.2.A.

Now we briefly sketch the rerouting algorithm proposed in [50], assuming $\sum_{k=1}^{K} d(k) = 1$ and $c = 1$ without loss of generality.

1. **Determine the initial feasible solution:** set $l(e) = 1/m$ for all edges $e \in E$. For each demand k, set $f(p) = d(k)$ for one shortest path p from the source to the sink. Set $f(e) = \sum_{p:e\in p} f(p)$ for all $e \in E$. And set $\sigma_0 = \frac{\epsilon^2}{32} m^3$.

2. **Compute new distance function and find shortest path:** Set
$$l(e) = \frac{e^{(2m^2/\epsilon)f(e)}}{\sum_{e'\in E} e^{(2m^2/\epsilon)f(e')}}$$
for all $e \in E$. And for each commodity k, find the shortest path distance $dist(k)$ under this distance function l.

Table II.1: Analogous Results in the Theories of the Single Commodity Flow and the Concurrent Flow

Properties	**Single Commodity Flow**	**Concurrent Flow**
Objective	Maximize value of source-to-sink flow	Maximize throughput of concurrent flow
Cut Inequality	Flow value less than or equal to capacity of any cut	Throughput less than or equal to density of any cut
Critical Saturated Edges	Edges saturated by every maximum flow contain a minimum cut	Edges saturated by every maximum concurrent constitute k-partite cut
Duality Theorem	The maximum value of flow equals the minimum capacity of cut	The maximum throughput of concurrent flow equals the minimum density of k-partite cut
Proof/Algorithm Paradigm	Flow augmentation proves duality and provides an efficient algorithm	Flow rerouting proves duality and provides an efficient algorithm

3. **Compute the throughput, distance upper bound, and check for termination:** Set $f^* = \max\{f(e)|e \in E\}$, $\lambda = 1/f^*$, $d = \frac{\sum_{e \in E} l(e)}{\sum_{k=1}^{K} dist(k)d(k)}$. If $(d - \lambda)/d \leq \epsilon$, output and halt.

4. **Reroute flow:** Find a particular active path p^k such that

$$d(p^k) - dist(k) = \max\{d(p^{k'}) - dist(k')|1 \leq k' \leq K, d(k)' \geq \sigma_0\} \qquad \text{(II.9)}$$

Let p_* be a particular shortest path between the source and sink of commodity k. Let the notation $p^k - p_*$ indicate those edges in p^k but not in p_*. Then set $l_1 = \sum_{e \in p^k - p_*} l(e)$ and $l_2 = \sum_{e \in p_* - p^k} l(e)$. Set

$$\sigma = \frac{\epsilon}{4m^2} log \frac{l_1}{l_2}$$

If $f(p^k) \geq \sigma + \sigma_0$, then reroute σ units of flow from p^k to p_*; otherwise, reroute all the flow on p^k to p_*. Go to step 2.

To explain the algorithm succinctly, step 1 supplies the flows to satisfy the demand; step 2 assigns much larger distances to the edges hosting larger flows, which makes the paths containing highly utilized edges less attractive for future flow assignments, since rerouted flow is always sent on a shortest distance path; step 3 simply checks the termination condition that primal and dual solutions are close enough; in step 4, the algorithm generally reduces the flow on high utilized edges and increases the flow on poorly utilized edges.

Regarding the complexity of this approximation algorithm, assuming there exist demands among all pairs of nodes in the graph, i.e. $k = n(n-1)/2$, step 2 and step 4 take $O(n^3)$ and $O(n|P|)$ in each iteration respectively, where P is the active path set. Actually, $|P| = O(m^3\epsilon^{-2})$ — the number of active paths is limited because of the σ condition in the algorithm — therefore each iteration takes the time $O(nm^3\epsilon^{-2})$. Furthermore, the authors prove that the total number of iterations is bounded by $O(m^4\epsilon^{-3})$. Hence the total complexity of the approximation algorithm is $O(nm^7\epsilon^{-5})$.

Following this innovative work, there were several papers that improved the complexity or extended the algorithm to general concurrent flow problem. The first improvement was by Klein et al. [35]. The algorithms they proposed still follow the same framework as that in [50], while the major modification is in Step 4 of the aforementioned algorithm, which involves the paths selected to reroute flows as well as the rerouted amount. In [50], the algorithm always chooses a flow path that has the largest difference from the shortest path between the source and the sink, as seen in Equation (11). In [35], the paths selected are not necessary to be so restricted. Later, Goldberg [23] simplified the randomized process and reduced the complexity of the randomized algorithm by ϵ^{-1} to $O(\epsilon^{-2}Knm \log k \log^3 n)$. Following their approach, Radzik [46] proved that choosing the commodities randomly can be replaced by choosing in the round-robin fashion, therefore complexity of the deterministic algorithm matches that of the randomized algorithm $O(\epsilon^{-2}Knm \log k \log^3 n)$.

The idea was continuously used in [45], but extended to solve more generalized problems. The flow rerouting algorithms achieve the best results by the incremental modifications of the following two papers: Karger and Plotkin [30] used the round-robin idea in [46] to modify the algorithms in [45] so that the deterministic algorithm has the bound $O^*(\epsilon^{-3}Kmn)$; Grigoriadis and Khachiyan [24] improved the approximation computations for the single minimum-cost flow problems and achieved the bound $O^*(\epsilon^{-2}Kmn)$. However, after that, the rerouting algorithms were replaced by augmenting algorithms, which we shall discuss in the next part.

II.2.B Flow Augmenting Algorithms

The flow augmenting algorithms were originated from [56]. In that paper, Young proposed *oblivious rounding algorithms* to solve general fractional packing and covering problems, which includes multi-commodity flow problems. Particularly, for the multi-commodity flow problems, Young's algorithm repeatedly augments one unit flow along a shortest path, where the length of an edge is initially 1 and is multiplied by $1 + \epsilon f(e)/u(e)$ each time the edge is used. The first paper that made use of the idea in [56] to derive combinatorial MCF approximation algorithms was due to Garg and Konemann [22]. In that paper, simple and fast algorithms were proposed for maximum multi-commodity flows, maximum concurrent flows, and minimum cost multi-commodity flows. Here we discuss the maximum concurrent flow algorithm in more detail.

As shown in the previous section, the dual objective of the maximum concurrent flow problem is equivalent to $D(l)/\alpha(l)$, where $D(l) = \sum_{e \in E} u(e)l(e)$ and $\alpha(l) = \sum_{k=1}^{K} d(k)dist(k)$, $dist(k)$ is the shortest path between the source and sink of demand k under the distance function l. The dual to the maximum concurrent flow problem can be viewed as an assignment of lengths to edges, such that $D(l)/\alpha(l)$ is minimized. Let β be the minimum.

The algorithm in [22] runs in t iterations till $D(t) \geq 1$. Each iteration we route the K commodities one by one. Particularly, $d(k)$ units of commodity k are routed in a sequence of steps. Let $l_{i,k}^{s-1}$ be the length function before the s^{th} step and let $P^s_{i,k}$ be the shortest path between s_k and t_k, the source and sink of demand k. In this step we route $f^s_{i,k} = \min\{u, d^s_{i,k}\}$ units of flow along $P^s_{i,k}$, where u is the capacity of the minimum capacity edge on this path. After routing, the length of each edge e in the path is modified as $l^s_{i,k} = l^{s-1}_{i,k}(e)(1 + \epsilon\frac{f^s_{i,k}}{u(e)})$. Then the amount to be routed is reduced by $f^s_{i,k}$. The iteration ends after p steps when all the $d(k)$ are routed.

Since the length function is monotonically increasing, after routing all flow of commodity k, we have

$$D(l^p_{i,k}) \leq D(l^0_{i,k}) + \epsilon \cdot d(k)dist_k(l^p_{i,k})$$

and after routing all commodities in the i^{th} iterations we have

$$D(l_{i,K}) \leq D(l_{i,0}) + \epsilon\sum_{k=1}^{K} d(k)dist_k(l_{i,k})$$

which is

$$D(i) \leq D(i-1) + \epsilon\alpha(i)$$

Since $\frac{D(i)}{\alpha(i)} \geq \beta$ we have

$$D(i) \leq \frac{D(i-1)}{1-\epsilon/\beta}$$

If the initial length of each edge is δ, then $D(0) = m\delta$. Then we can derive

$$D(i) \leq \frac{m\delta}{(1-\epsilon/\beta)^i} \leq \frac{m\delta}{1-\epsilon}e^{\frac{\epsilon(i-1)}{\beta(1-\epsilon)}}$$

The procedure stops at $D(t) \geq 1$, therefore,

$$1 \leq D(t) \leq \frac{m\delta}{1-\epsilon}e^{\frac{\epsilon(t-1)}{\beta(1-\epsilon)}}$$

which implies

$$\frac{\beta}{t-1} \leq \frac{\epsilon}{(1-\epsilon)ln\frac{1-\epsilon}{m\delta}}$$

Furthermore, the primal solution λ after $t-1$ iterations can be proven to be greater than $\frac{t-1}{\log_{1+\epsilon} 1/\delta}$. Consider an edge e, for every $u(e)$ units of flow routed through e, we increase its length by at least a factor $1+\epsilon$. Initially, its length is $\epsilon/u(e)$ and after $t-1$ iterations, since $D(t-1) < 1$, the length of e satisfies $l_{t-1,k}(e) < 1/u(e)$. Therefore the total amount of flow through e in the first $t-1$ iterations is strictly less than $\log_{1+\epsilon} \frac{1/u(e)}{\delta/u(e)} = \log_{1+\epsilon} 1/\delta$ times its capacity. Scaling the flow by $\log_{1+\epsilon} 1/\delta$ implies the claim. Then the duality gap β/λ can be proven to be bounded by $(1-\epsilon)^{-3}$. And, by the weak duality theorem, the authors show that the total number of steps is at most $(2K \log K + m)\lceil \frac{1}{\epsilon} \log_{1+\epsilon} \frac{m}{1-\epsilon} \rceil$ and each of these involves one shortest path computation.

However, the above proof is only valid when assuming $\beta \geq 1$. If $\beta < 1$, the demands need to be scaled up in order to make $\beta \geq 1$. The estimation of β requires to compute K maximum single-commodity flows. Hence, the overall complexity is $O^*(\epsilon^{-2}m(m+K) + K$ max flows).

After this innovative work, there are two papers later on that incrementally improve the complexity by following the framework. Fleischer [20] noticed that for the maximum multi-commodity flow problem, one can use $(1+\epsilon)$ approximate shortest paths, in order to avoid recalculations of shortest paths for commodities with a common source. Therefore the complexity for the maximum multi-commodity flow problem is reduced from $O^*(\epsilon^{-2}Km^2)$ to $O^*(\epsilon^{-2}m^2)$, which is independent of the number of commodities. Unfortunately, this technique does not apply to the maximum concurrent flow. Nevertheless, with a smarter way to estimate the value β, the computation of K maximum single-commodity flows is greatly reduced; the complexity to calculate the maximum concurrent flow is reduced to $O^*(\epsilon^{-2}m(m+K))$. Recently, Karakostas [29] further improved the maximum concurrent algorithms. He found that in each step, we can route the commodities which have the same source together, without affecting the convergence. Hence the complexity is further improved to $O^*(\epsilon^{-2}m^2)$, which is independent of the number of commodities K. This is the current best approximation algorithm for the maximum concurrent flow problem.

In addition, these three papers handle the minimum-cost multi-commodity flow in the similar way: given a cost budget B, the algorithms can be converted to handle the maximum concurrent flow with cost budget. And then the optimal cost can be found by binary search with a $\log M$ factor, where M is the largest number used to specify capacities, demands or costs.

II.3 Algorithms

II.3.A Overview

This section will introduce the MCF algorithms we propose to perform the interconnection synthesis and optimization. Usually, the interconnection networks are modeled using the graph, the communications are represented by commodities, the wiring area is modeled using capacity constraints, and the latency and power are usually reflected as edge costs. Our algorithms are based on the approximation schemes proposed in [20] and [29]. However, our algorithms are differentiated from the previous works in the following aspects:

1. The capacity constraints are imposed on a bundle of edges instead of one edge. This new constraint is raised because in the interconnection synthesis, a bunch of wires usually share a common wiring/routing area. We propose an approximation scheme based on the one proposed in [29] and prove the convergence.

2. In interconnection network synthesis, instead of the global average latency, sometimes we do care the individual latency of several communications (commodities). Thus, reflected in the formulation, each individual commodity has a specific cost constraint. We show that the scheme is applicable with this extension.

3. We also propose an interval estimation technique to speed up the binary search process when solving minimum cost MCF problems. Although it is a heuristic method, we prove that it does not affect the accuracy of the produced solutions, while significantly improves the speed.

In the following three subsections, we shall introduce the baseline algorithms with general capacity constraints, the algorithms with individual commodity constraints, and the interval estimation method respectively.

II.3.B Baseline MCF Algorithms

First, our baseline algorithm finds the largest λ such that there is a multicommodity flow which routes at least λd_i units of commodity i, where each edge is associated with a triple of unit cost (A_e, P_e, D_e) (which represent the unit width, unit power, and unit delay of a wire respectively in the real world context). A is the capacity constraint which is imposed on a set of edges $S(q)$, and PW and LT are the total power and latency constraints. Hence, the problem formulation can be described as follows.

$$\begin{array}{rc}
Primal: & \\
Max: & \lambda \\
\forall j: & \sum_{p\in p_j} f(p) \geq \lambda d_j \\
\forall q: & \sum_{e\in S(q)} A_e \sum_{p:e\in p} f(p) \leq A \\
& \sum_{i=1}^{k} \sum_{p\in p_i} \sum_{e\in p} f(p)P_e \leq PW \\
& \sum_{i=1}^{k} \sum_{p\in p_i} \sum_{e\in p} f(p)D_e \leq LT \\
\forall p: & f(p) \geq 0
\end{array}$$

The following is the dual problem. Besides a variable X_e for each grid area constraint and a variable Z_j for every commodity demand constraint, the dual problem has another two variables, ϕ_p corresponding to the power budget constraint, and ϕ_d corresponding to the latency budget constraint:

$$
\begin{aligned}
&Dual: \\
&Min: && A\textstyle\sum_{q=1}^{n} X_q + PW\phi_p + LT\phi_d \\
&\forall j, \forall P \in \rho: && \textstyle\sum_{e\in P} A_e \sum_{e\in S(q)} X_q + \sum_{e\in P} P_e\phi_p \\
& && + \textstyle\sum_{e\in P} D_e\phi_d \geq Z_j \\
& && \textstyle\sum_{j=1}^{k} d_j Z_j \geq 1 \\
&\forall q: && X_q \geq 0 \\
&\forall j: && Z_j \geq 0
\end{aligned}
$$

Assume the subroutine $mcf(G, d, LT, PW)$ could return such a λ, the power minimization MCF algorithm finds the minimum power that satisfying $\lambda \geq 1$ by recursively binary search, as shown in Algorithm 1, where we use λ_{max} to denote the concurrent value without power budget constraint, i.e. $PW = \infty$.

Algorithm 1 Power Minimization MCF Algorithm

```
1: Input: graph $G$, demand $d$, latency constraint $LT$, threshold $\epsilon$
2: Output: $(1+\epsilon)$ optimal power
3: set $\lambda_{max} \leftarrow mcf(G, d, LT, \infty)$
4: set lower bound $lb \leftarrow 0$
5: upper bound $ub \leftarrow$ total power under $\lambda_{max}$
6: while $(ub - lb)/ub > \epsilon$ do
7:    $\lambda \leftarrow mcf(G, d, LT, (lb + ub)/2)$
8:    if $\lambda \geq 1$ then
9:       $ub \leftarrow (lb + ub)/2$
10:      else $lb \leftarrow (lb + ub)/2$
11:   end if
12: end while
13: Output $ub$
```

$mcf(G, d, LT, PW)$ subroutine iteratively updates the primal and dual values till the gap is small enough. The primal value λ is updated by adjusting the flows. To calculate dual values, we define edge length as:

$$l(e) := A_e \sum_{q:e \in S(q)} X_q + PW \cdot \phi_p + LT \cdot \phi_d \tag{II.10}$$

So dual is equivalent to:

$$Min : \frac{A \sum_{q=1}^{n} X_q + PW\phi_p + LT\phi_d}{\sum_{j=1}^{k} d_j \cdot dist(j)} \tag{II.11}$$

where $dist(j)$ is the shortest path from the source to the sink of commodity j under the length function $l(e)$. The process is described in Algorithm 2.

Algorithm 2 proceeds in phases and each phase is composed of k iterations. In iteration j of the i^{th} phase we route d_j units of commodity j in a sequence of steps. In each step, a shortest path P from source s_j to sink t_j is computed using the current length function. The dual variables X_q are updated as

$$X_q = X_q(1 + \frac{\delta}{3} \cdot \frac{\sum_{e \in S(q)} A_e f(e)}{A}) \tag{II.12}$$

and ϕ_p and ϕ_d are updated in the similar fashion.

Regarding the convergence of Algorithm 2, by carefully choosing the initial values X_0, we have the following theorems:

Theorem 1 *When the algorithm terminates,* $\frac{\lambda}{D} \geq 1 - \delta$.

Theorem 2 *The algorithm runs in* $O(\delta^{-2}|E|^2)$.

Theorem 1 guarantees the $(1 - \delta)$ optimality and Theorem 2 shows the efficiency. The proofs are similar to those in [22] and [29]. However, the formulations in the previous works all treat the capacity constraints are on the edges; while here, the constraints are on a set of edges. Therefore to update the dual variables, we need to modify the original formula in [22] and [29]

$$l(e) = l(e)(1 + \frac{\delta}{3} \cdot \frac{f(e)}{c(e)}) \tag{II.13}$$

Algorithm 2 $(1-\delta)$ Maximum Concurrent Flow Algorithm

```
1: Input: graph $G$, demand $d$, latency constraint $LT$, power budget $PW$, threshold $\delta$
2: Output: $(1-\delta)$ optimal maximum concurrent value $\lambda$
3: $\forall q$, set $f(e) \leftarrow 0$, $X_q \leftarrow X_0$, $\phi_p \leftarrow X_0$, $\phi_d \leftarrow X_0$
4: $l(e) \leftarrow A_e \sum_{q:e \in S(q)} X_q + PW \cdot \phi_p + LT \cdot \phi_d$
5: while $A \sum_{q=1}^{n} X_q + PW\phi_p + LT\phi_d \leq 1$ do
6:    for each commodity $j$ do
7:       $rd_j \leftarrow d_j$
8:       while $rd_j > 0$ do
9:          Route $f$ units of flow from $s_i$ to $t_j$ along the shortest path $P$
10:         $f(e) \leftarrow f(e) + f$, $\forall e \in P$
11:         $X_q \leftarrow X_q(1 + \frac{\delta}{3} \cdot \frac{\sum_{e \in S(q)} A_e f(e)}{A})$
12:         $\phi_p \leftarrow \phi_p(1 + \frac{\delta}{3} \cdot \frac{\sum power}{PW})$, $\phi_d \leftarrow \phi_d(1 + \frac{\delta}{3} \cdot \frac{\sum latency}{PW})$
13:         $l(e) \leftarrow A_e \sum_{q:e \in S(q)} X_q + PW \cdot \phi_p + LT \cdot \phi_d$
14:         $rd \leftarrow rd - f$
15:      end while
16:   end for
17:   compute primal $\lambda$ by scaling down all $f(e)$ subject to area, power and latency constraints
18:   compute dual $D \leftarrow \frac{A \sum_{q=1}^{n} X_q}{\sum_{j=1}^{k} d_j \cdot dist(j)}$
19: end while
20: return $\lambda$
```

where $f(e)$ is the total flow on edge e and $c(e)$ is the edge capacity, to the one in Equation (II.12). This is from the intrinsic spirit of the dual variable updating scheme: the dual variables reflect the congestion level of the edge or a set of edges, therefore we always update it using the ratio of the flow versus the total available resource. In addition, the power and latency constraints can be viewed as two "pseudo edges" with capacities PW and LT, so ϕ_p and ϕ_d have the similar update formula.

It is worth noting that the dual variables X_q are associated with a set of edges instead of a single edge, therefore we need to apply formula (II.14) to further compute the edge lengths. Sometimes this results complicated cases. Refer to Figure II.3, consider a path consisting of two edges (a, e) and (e, c), the lengths should be updated as

$$l((a, e)) = A \cdot (X_1 + X_2 + X_3 + X_4) + W\phi_p + L\phi_d$$

$$l((e, c)) = A \cdot (X_3 + X_4) + W\phi_p + L\phi_d$$

Note that X_3 and X_4 do need to be counted twice in the path, since the path crosses them twice and each time the flow contributes to the congestion individually. This issue is carefully handled in the implementation.

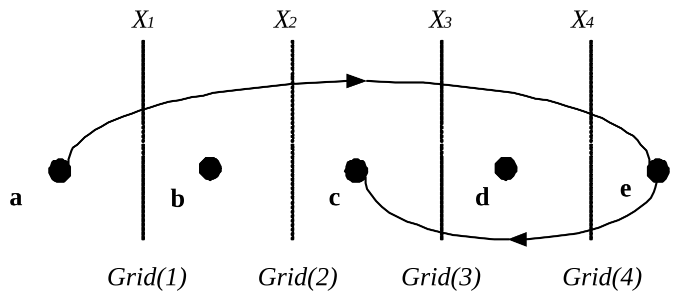


Figure II.3: Length Function on Edge

II.3.C MCF Algorithms with Pairwise Latency Constraints

This section discusses a variation of the MCF problem where the latency constraint is imposed on each individual commodity. Let LT_i represents the latency constraint on commodity i, the problem formulation is as follows:

$$\begin{aligned}
&Primal: && \\
&Max: && \lambda \\
&\forall j \in [1,k]: && \textstyle\sum_{p \in p_j} f(p) \geq \lambda d_j \\
&\forall q: && \textstyle\sum_{e \in S(q)} A_e \sum_{p: e \in p} f(p) \leq A \\
& && \textstyle\sum_{i=1}^{k} \sum_{p \in p_i} \sum_{e \in p} f(p) P_e \leq PW \\
&\forall i \in C: && \textstyle\sum_{p \in p_i} f(p) \sum_{e \in p} D_e \leq LT_i \\
&\forall p: && f(p) \geq 0
\end{aligned}$$

The following is the dual problem. Besides that X_e, Z_j, ϕ_p keep the same meaning, each individual critical commodity i has a dual variable Y_i corresponding to the latency budget constraint on that commodity:

$$\begin{aligned}
&Dual: && \\
&Min: && A \textstyle\sum_{q=1}^{n} X_q + PW \phi_p + \sum_{i \in C} LT_i Y_i \\
&\forall j, \forall P \in \rho: && \textstyle\sum_{e \in P} A_e \sum_{e \in S(q)} X_q + \sum_{e \in P} P_e \phi_p \\
& && \textstyle + \sum_{e \in P} D_e Y_j \geq Z_j \\
& && \textstyle\sum_{j=1}^{k} d_j Z_j \geq 1 \\
&\forall q: && X_q \geq 0 \\
&\forall j: && Z_j \geq 0 \\
&\forall i: && Y_i \geq 0
\end{aligned}$$

The algorithms to solve the MCF problem with pairwise latency constraints share the same ideas with the aforementioned algorithms, except for the edge length. Since there is an array of latency constraints on a group of critical commodities, now each edge has an array of lengths instead of only one edge length. When flow of a certain critical commodity goes through an edge, only the edge length that corresponds to the specific critical commodity will be used and updated for shortest path searching. The following is the length function for $l(e)_i$ on edge e.

$$l(e)_i := A_e \sum_{q:e \in S(q)} X_q + P_e \phi_p + D_e Y_i \tag{II.14}$$

And the update function for dual variable Y_i is:

$$Y_i = Y_i(1 + \frac{\delta}{3} \cdot \frac{\sum_e D_e f(e)}{LT_i}) \tag{II.15}$$

II.3.D Interval Estimation Heuristic

In Section II.3.B, while Algorithm 1 needs to obtain MCF solutions with $(1+\epsilon)$ optimal power values, Algorithm 2 returns us $(1-\delta)$ optimal concurrent flow. Therefore the values of ϵ and δ are associated "pseudo polynomially": δ has to be determined by both the value of ϵ and the unit edge cost P_e, which leads to extremely slow convergence in some pathological cases.

Thus, we propose a heuristic interval estimation technique to speedup the process. The idea is to estimate the new lower bound lb' and upper bound ub' while performing the approximation algorithms, and terminate the search once $ub' - lb' \leq (ub - lb)/2$ in each step of the binary search scheme.

We define a function monotonically increasing $P(\lambda)$, where λ is the concurrent flow and $P(\lambda)$ is the minimum power under this concurrent flow (therefore $P(1)$ is the target optimal value). The curve is shown in Figure II.4. Furthermore, we have the following lemma:

Lemma: $P(\lambda)$ is a convex function.

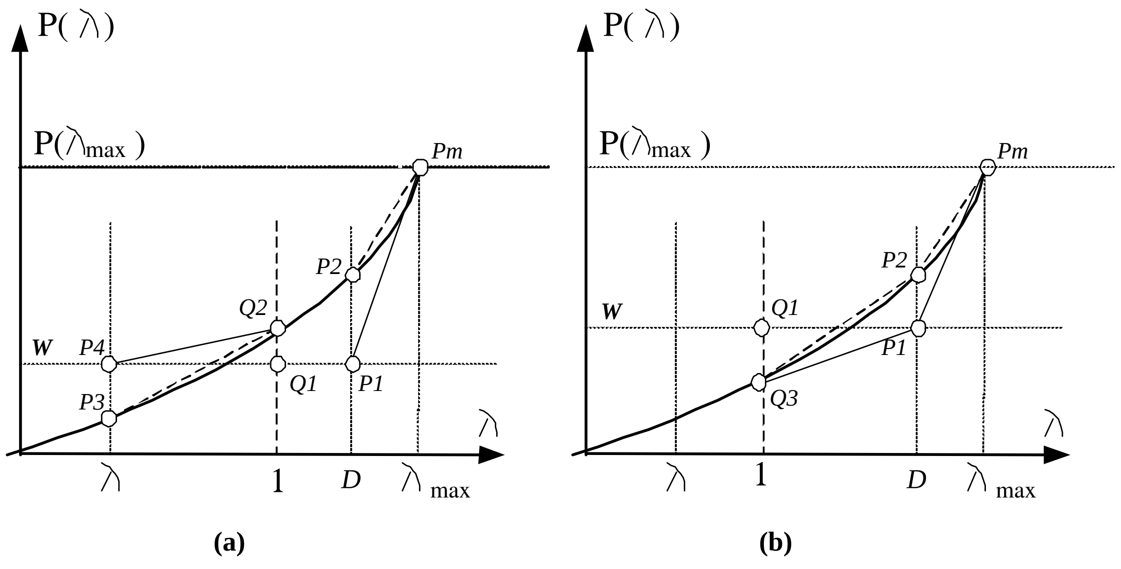


Figure II.4: Interval Estimation

Proof: For a specific λ_1, the minimum power should be $P(\lambda_1)$; scaling down all the flows by half, the concurrent flow would be $\frac{\lambda_1}{2}$, and the power is $\frac{P(\lambda_1)}{2}$. On the other hand, when the concurrent flow is $\frac{\lambda_1}{2}$, the minimum power should be $P(\frac{\lambda_1}{2})$, therefore we have $P(\frac{\lambda_1}{2}) \leq \frac{P(\lambda_1)}{2}, \forall \lambda_1 \leq \lambda_{max}$. So the function is convex. □

We use the following theorem to estimate the lower bound lb' and upper bound ub':

Theorem 3 *Given a feasible primal value λ and a feasible dual value D under the power budget PW, we have*

$$PW - s \cdot (D - 1) \leq P(1) \leq PW + s \cdot (1 - \lambda) \tag{II.16}$$

where $s = \frac{P(\lambda_{max}) - PW}{\lambda_{max} - D}$. Hence, $lb' \leftarrow \max\{lb', PW - s \cdot (D - 1)\}$, $ub' \leftarrow$ $\min\{ub', PW + s \cdot (1 - \lambda)\}$

We sketch the proof for $P(1) \leq PW + s \cdot (1 - \lambda)$ here. Refer to Figure II.4 (a), let the lines $x = \lambda$, $x = 1$, $x = D$ and $x = \lambda_{max}$ intersect the function curve at P_3, Q_2, P_2 and P_m, and $x = 1$, $x = 1$ and $x = D$ intersect $y = PW$ at P_4, Q_1 and P_1 respectively (we use x and y to denote the two axes). We then have

$$P(1) = PW + S_{P_4Q_2} \cdot (1 - \lambda) \tag{II.17}$$

where $S_{P_4Q_2}$ is the slope of the line P_4Q_2. And, it is easy to identify that $S_{P_4Q_2} \leq S_{P_3Q_2} \leq S_{P_2P_m} \leq S_{P_1P_m}$, by the property of the convex function. And since $s = S_{P_1P_m}$, we have $P(1) \leq PW + s \cdot (1-\lambda)$. Similarly, $PW - s \cdot (D-1) \leq P(1)$ can be proven by the similar approach, as shown in Figure II.4 (b). □

Consider in a certain iteration of the approximation algorithm, we have a feasible primal value λ and a feasible dual value D, where $\lambda < 1$ and $D > 1$ (otherwise the process will terminate). To approximate the range of the optimal solution $P(1)$, We need to consider two cases:

1. When $P(1) \geq PW$, as shown in Figure II.4 (a), then line $x = 1$ intersects $y = PW$ and the function curve at Q_1 and Q_2 respectively (we use x and y to denote the two axes). Then $P(1) = PW + |Q_1Q_2|$. Assume $x = \lambda$ intersects the function curve and $y = PW$ at P_3 and P_4 respectively, we can easily compute $|Q_1Q_2| = S_{P_4Q_2} \cdot |P_4Q_1| = S_{P_4Q_2} \cdot (1 - \lambda)$,where $S_{P_4Q_2}$ is the slope of the line P_4Q_2. Then,

$$P(1) = PW + S_{P_4Q_2} \cdot (1 - \lambda) \tag{II.18}$$

Unfortunately, we do not know the exactly value of $S_{P_4Q_2}$; therefore we use the following method to approximate the slope:

We know that the primal feasible solution λ has the power at most PW, consequently $P(\lambda) \leq PW$, since $P(\lambda)$ denotes the optimal power value. Therefore P_3 is below P_4 (or equal), and then $S_{P_3Q_2} \geq S_{P_4Q_2}$. Let $x = D$ intersect $y = PW$ and the function curve at P_1 and P_2, and $x = \lambda_{max}$ intersect the function curve at P_m, we can conclude that $S_{P_2P_m} \geq S_{P_3Q_2}$, since all the four points are in the curve and the function is convex. Finally, $S_{P_1P_m} \geq S_{P_2P_m}$ since P_1 is below P_2 (or equal), which are symmetric to P_3 and P_4. Substitute the inequalities to Equation (II.18), we have

$$P(1) \leq PW + S_{P_1P_m} \cdot (1 - \lambda) \tag{II.19}$$

2. When $P(1) < PW$, as shown in Figure II.4 (b), using the similar derivation, we can get

$$P(1) = PW - S_{Q_3P_1} \cdot (D-1) \tag{II.20}$$

And because $S_{P_1P_m} \geq S_{P_2P_m} \geq S_{P_1P_2} \geq S_{P_1Q_3}$, we obtain that

$$P(1) \geq PW - S_{P_1P_m} \cdot (D-1) \tag{II.21}$$

In fact, inequalities (II.19) and (II.21) hold in both cases, as in the first case, $P(1) \geq PW$ and in the second case $P(1) < PW$. Since $S_{P_1P_m}$ can be easily computed as $\frac{P(\lambda_{max})-PW}{\lambda_{max}-D}$, we can compute the new lower bound lb' and upper bound ub' as:

$$lb' = PW - S_{P_1P_m} \cdot (D-1) \tag{II.22}$$

$$ub' = PW + S_{P_1P_m} \cdot (1-\lambda) \tag{II.23}$$

According to Theorem 3, Algorithm 1 and Algorithm 2 can be improved as Algorithm 3 and Algorithm 4. The new algorithms run much faster than the original ones, but the accuracy is not compromised.

Algorithm 3 Modified Power Minimization MCF Algorithm

```
1: Input: graph $G$, demand $d$, latency constraint $LT$, threshold $\epsilon$
2: Output: $(1+\epsilon)$ optimal power
3: As in Algorithm 1 Steps 3–5
4: while $(ub-lb)/ub > \epsilon$ do
5:     $(lb', ub') \leftarrow mcf(G, d, LT, (lb+ub)/2)$
6:     $lb \leftarrow lb'$; $ub \leftarrow ub'$
7: end while
8: Output $ub$
```

II.4 Discussion

In this chapter, we discussed the algorithms for MCF problems, which are the core of our design flow for interconnection network synthesis and optimization.

Algorithm 4 Modified Maximum Concurrent Flow Algorithm

```
1:  **Input:** graph $G$, demand $d$, latency constraint $LT$, power budget $PW$, threshold $\delta$
2:  **Output:** new lower bound $lb'$ and upper bound $ub'$
3:  As in Algorithm 2 Steps 3–4
4:  $lb' \leftarrow lb$; $ub' \leftarrow ub$
5:  **repeat**
6:    As in Algorithm 2 Steps 6–18
7:    $lb' \leftarrow \max\{lb', PW - s \cdot (D-1)\}$
8:    $ub' \leftarrow \min\{ub', PW + s \cdot (1-\lambda)\}$
9:    **if** $ub' - lb' \leq (ub - lb)/2$ **then**
10:     **return** $(lb', ub')$
11:   **end if**
12: **end repeat**
```

Although there have been a lot of approaches in literature in this classic problem, we explored the fully polynomial approximation schemes due to its accuracy and also efficiency. We augmented the existing algorithms by incorporating more practical constraints and also improving the running speed. This chapter built the theoretical foundation of the dissertation.

Regarding the MCF algorithms, however, there are still a lot of things to be explored in the theoretical aspects. We highlight a few of them as follows:

1. Currently, exponentiation functions like $(1 + \epsilon\frac{f(e)}{u(e)})$ updates are used in the flow augmenting algorithms (it is indeed very similar to exponentiation functions). The choices of these functions facilitate the analysis proof but there are no evidence to show how the theoretical goodness of these functions. If we could build the relation between the length function and the convergence rate (also the algorithm complexity), experimental work would be reduced and the performance would be improved by choosing the suitable length function.

2. The parameter ϵ controls the convergence speed: the larger ϵ is, the larger difference between more congested edges and less congested ones. One one hand, larger ϵ may achieve faster convergence if it increases lengths on the correct critical edges; on the other hand, larger ϵ may update on the wrong (non-critical) edges so that more troubles would be created. Under this circumstance, smaller ϵ is preferred to adjust the flows and lengths more smoothly to avoid oscillations. The existing algorithms all use a constant ϵ all the way. In the experiments we conduct, however, a dynamic ϵ can achieve a better performance: a larger ϵ is used in the beginning to quickly identify critical edges and smaller ϵ values are used later to adjust the flows and lengths in a more smooth and accurate manner. But we have not figured out the optimal way to dynamically change ϵ so that the fastest convergence can be guaranteed.

3. In the current analysis, the duality gap 3ϵ can be guaranteed when the updating parameter is ϵ. But the experimental experience tells us that normally the gap can be much smaller than 3ϵ. A tighter bound is important in practice because we could use a larger ϵ to reach the same accuracy therefore the convergence could be faster. Observing that the existing analysis is done by assuming extreme cases in quite a few places, we guess that a tighter analysis may be possible.

III

On-chip Interconnect Synthesis and Optimization

III.1 Overview

In this chapter, we focus on structured NoC design in multicore architectures. In a structured multicore architecture, processing cores are arranged on chip in a regular manner, and each core contains a router for communicating with other cores. Structured multicore architectures have been proposed and implemented by various research efforts, such as MIT RAW architecture [51], and UT Austin TRIPS architecture [32], etc.

In the previous work [28], Hu *et al.* proposed an MCF based scheme to optimize NoC network topology and interconnect circuit styles simultaneously. However, none of the previous works considered power and latency as the design objectives simultaneously and studied their relations. In this chapter, we propose a design methodology that selects NoC topologies and the corresponding interconnect wire styles, so that power consumption is minimized subject to average communication latency constraints. With the improved MCF approximation algorithm, we are able to search for the optimal NoC designs with size up to 12×12. Our main contributions are as follows:

- We introduce a variety of wire styles into NoC design, and incorporate latency constraints and power minimization objectives into a unified MCF model, with simultaneous optimization on network topologies, physical embedding, and interconnect wire styles. For any given average node to node communication latency requirement, our algorithm finds the best NoC implementations. Experiments show that for 8×8 NoC, an optimized design can improve the power-latency product by up to 52.1%, 29.4% and 35.6%, if compared with mesh, torus and hypercube topologies, respectively. Furthermore, by sacrificing 2% of latency constraints, power consumption of that optimized design can be improved by up to 19.4%. Carefully balancing between NoC power efficiency and communication latency is important in NoC design.

- We generate a wide range of NoC topologies. The generated topologies not only include most of the currently studied classic network topologies, such as two-dimensional mesh and torus, high-dimensional mesh and torus [54], hypercube, octagon [31], and twisted cube, etc., but also extend far beyond the range of these popular NoC topologies.

- We implement the MCF solver using approximation algorithms, which are significantly faster than the commercial linear programming solver CPLEX. We further optimize the solver by applying the interval estimation technique. Experiments show that this heuristic optimization can improve the convergence time by more than 300 times for NoC of size 7x7.

The rest of this chapter is organized as follows: Section III.2 will formulate the latency aware low power NoC synthesis problem. Section III.3 will briefly describe our design methodology. In Section III.4 we will give the experiment results. The summary of this chapter will be given in Section III.5.

III.2 Problem Statement

We assume a regular tile based structured NoC with $n \times n$ tiles. Each tile consists of a *processing core* and a *router*. Each tile can be regarded as a grid with certain area and dimension, and the total wiring area across a grid cannot exceed grid dimension. Each network link can be implemented with multiple wire styles, which have varying properties in terms of their area usage, power efficiency, and signaling latency.

Figure III.1 shows an example of a 4×4 tile based structured NoC. The network is linked by various types of wire styles with different capacities (Figure III.1a). The topology is a folded torus (Figure III.1b).

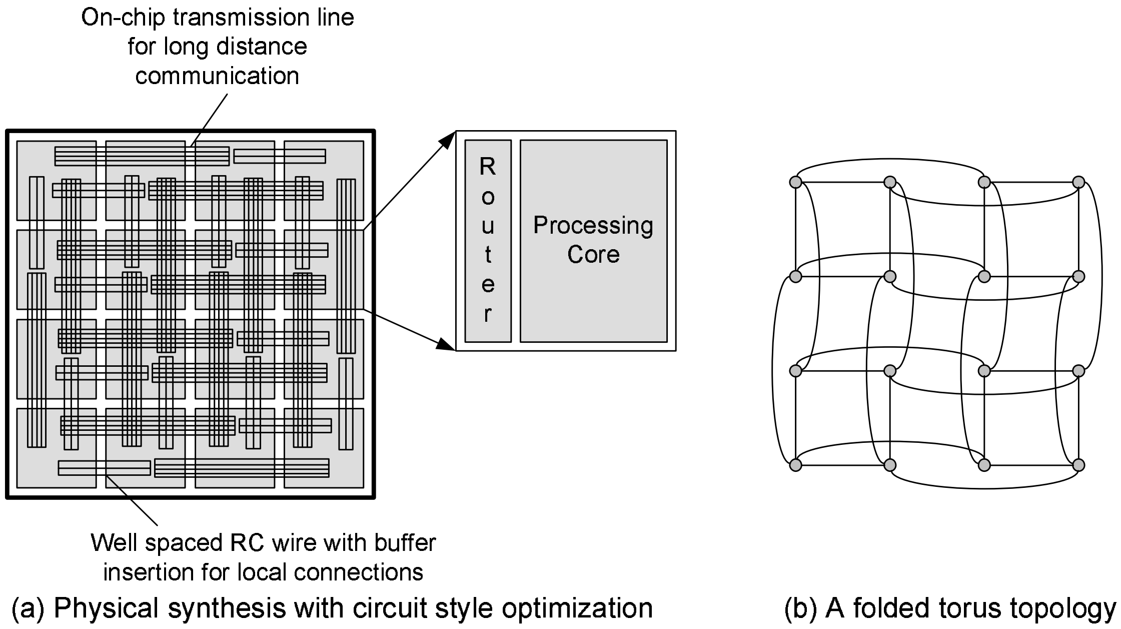

On-chip transmission line for long distance communication
R o u t e r
Processing Core
Well spaced RC wire with buffer insertion for local connections
(a) Physical synthesis with circuit style optimization
(b) A folded torus topology

Figure III.1: Tile-based Noc Architecture with Wire Style Optimization

We adopt source routing algorithms to route packets over NoC. Packets belonged to the same message can be routed through different paths to the destination tile, where software mechanisms take care of resembling packets together to rebuild the original message according to packet header information. When routing data packets to adjacent tiles, routing mechanism may dispatch packets to different types of wires according to their wire capacities. Since we use network flow models to formulate the NoC synthesis problem, our work is within the static flow scope. For applications with dynamic communications, queuing mechanisms and flow control mechanisms are needed to resolve the network contentions, and detailed simulations are required to observe the network dynamic behaviors. Though detailed simulation is a good approach for accurate network behavior estimation, it suffers from long evaluation time and high development efforts. On the contrary, though our static network flow model approach bypasses these important network design issues and loses some accuracy, it has merits of providing quick evaluation and useful guidance in the initial stages of NoC design.

In our work, we define an NoC topology graph as a directed graph $G = (V, E)$, where, each node $v_i \in V$ represents a tile, and each edge $e_{i,j} \in E$ represents a point to point interconnection between tile i and tile j.

An implementation of an NoC topology is a mapping from each edge to a particular wire style, $T(e) : E \rightarrow S$, and a mapping from each edge to the amount of wiring resources assigned to that edge, $C(e) : E \rightarrow R^{+}$. A particular wire style has three properties associated with it: the per edge routing area usage, the per bit energy, and the wire delay.

We formulate the problem as the following *communication latency aware minimum power NoC synthesis problem*:

Latency-constrained minimum power NoC synthesis problem:

We have an $n \times n$ array of tiles, a library of interconnect wire components of certain lengths:

Input: *The communication demand matrix between each pair of tiles*

Output: *The most power-efficient NoC topology $G = (V, E)$, and its physical implementation $T(e), C(e), \forall e \in E$*

Constraints: *(1) The communication latency requirements are satisfied; (2) The cross section wiring area can not exceed the grid dimension.*

III.3 Design Methodology

As shown in Figure III.2, for a structured $n \times n$ NoC, we first automatically generate the topology library. Then, based on power and delay libraries, we use an MCF model to evaluate latency constrained NoC power consumption for each topology in the topology library.

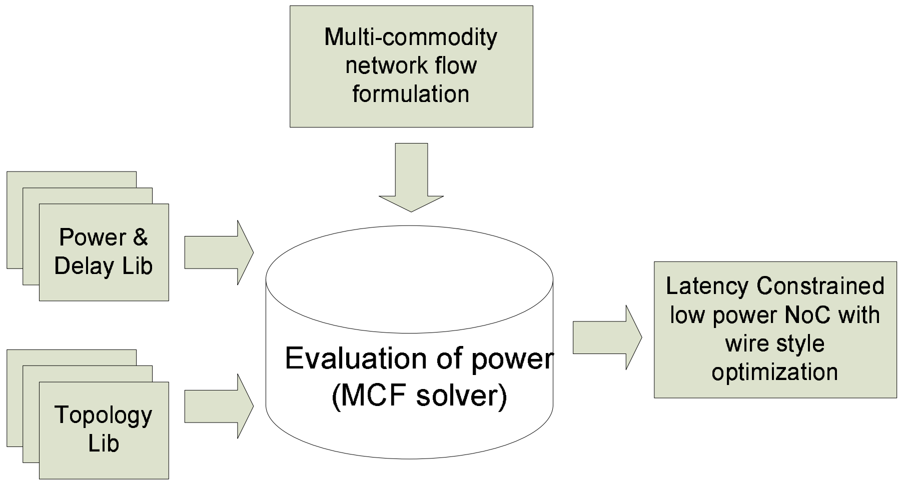

Multi-commodity network flow formulation
Power & Delay Lib
Topology Lib
Evaluation of power (MCF solver)
Latency Constrained low power NoC with wire style optimization

Figure III.2: Design Flow

In the following subsections, we describe our latency constrained low power MCF model, topology generation, and power and delay models, respectively. The MCF solver has been introduced in detail in Chapter II.

III.3.A Latency Constrained Minimum Power MCF Formulation

Since many notations are used in MCF formulations and approximation MCF algorithms in the rest of the chapter, Table III.1 summarizes them for quick reference.

For a given NoC topology graph $G = (V, E)$, we construct a flow graph. For each link between any two nodes, it consists of t edges, where t is the number of candidate wire styles of the link. Figure III.3 shows an example flow graph. There

Table III.1: Description of Symbols

Symbol	**Description**
V	Node set
E	Edge set
G	Graph, $G = (V, E)$
k	Number of commodities
d_i	Communication demand of commodity i
s_i	Source node of commodity i
t_i	Sink node of commodity i
p	A path in G
p_i	Set of paths for commodity i
$f(p)$	Flow amount on path p
$\mathbf{p}$	Set of paths for all commodities, $\mathbf{p} := \cup_i p_i$
$Grid(q)$	Set of all vertical or horizontal dimensions
A	Area constraint on grid dimension
P_w	Per-bit wire energy on a certain edge
P_r	Per-bit router energy on a certain edge
P_e	Per-bit total energy on edge e, $P_e = P_w + P_r$
D_w	Wire delay on a certain edge
D_r	Router delay on a certain edge
D_e	Total delay on edge e, $D_e = D_w + D_r$
A_e	Routing area usage on edge e
LT	Global latency bound
PW	Global power consumption bound
C	Set of critical paths
LT_i	Pairwise latency bound on commodity i
λ	Fraction of communication demand
ϵ	Accuracy of approximation MCF algorithm
X_q	Dual variable for each grid dimension
ϕ_p	Dual variable for global power consumption
ϕ_d	Dual variable for global latency
Z_j	Dual variable for communication demand
$l(e)$	Edge length regardless critical commodity
$l(e)_i$	Edge length for critical commodity i

are t edges from node v_m to node v_n. Each edge e is associated with the wire style (P_e, A_e, D_e). P_e, A_e, and D_e are the per-bit energy, routing area usage, and wire delay of edge e, respectively. The estimation of P_e and D_e will be described in Subsection III.3.C. A_e is measured for an edge with unit capacity.

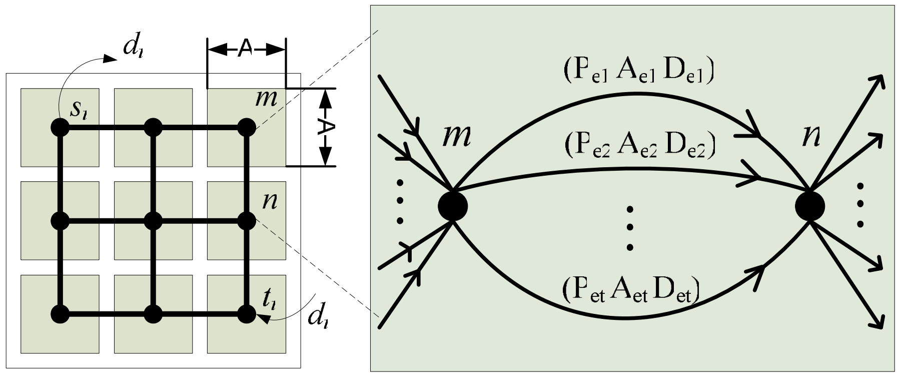

d_i
A
s_i
m
A
n
t_i
d_i
$(P_{e1}\ A_{e1}\ D_{e1})$
$(P_{e2}\ A_{e2}\ D_{e2})$
$(P_{et}\ A_{et}\ D_{et})$
m
n

Figure III.3: Flow Graph with Wire Style Optimization

Assume there are k commodities among all pairs of nodes. For each commodity i, which starts at node s_i and ends at node t_i, we are given a communication demand $d_i > 0$ which is the required bandwidth. We define critical commodities as those commodities, which have timing requirements on latencies between source and sink nodes. Assume C is the set of all critical commodities. For each vertical or horizontal dimension $Grid(q)$, we are given a grid dimension constraint A so that the sum of all the edge width on this grid is no more than A. Let p_i be the set of paths for commodity i, and let $\mathbf{p} := \cup_i p_i$. Variable $f(p)$ denotes the amount of flow sent along path p, for every $p \in \mathbf{p}$.

In our work, we study two types of communication latency constraints in NoC design. One is global average latency constraint, which is suitable for those applications without critical path timing requirements. Therefore, we assume the overall latency, which is the sum of the latencies for each flow, to be bounded by LT. So the average latency is $LT/\sum_{i=1}^{k} d_i$. On the other hand, for applications that have tight timing requirements on critical commodities, we set latency constraints to each specific critical commodity. We call this case pairwise average latency constrained low power NoC synthesis problem. We assume LT_i is overall latency constraint on individual commodity i, so the average latency is LT_i/d_i. We have formulations for each of the above two scenarios, respectively.

The following is the MCF formulation for global average latency constrained minimum power NoC synthesis.

$$Min: \quad \sum_{j=1}^{k} \sum_{p \in p_j} \sum_{e \in p} f(p) \cdot P_e \tag{III.1}$$

$$s.t. \quad \sum_{j=1}^{k} \sum_{p \in p_j} \sum_{e \in p} f(p) \cdot D_e \leq LT \tag{III.2}$$

$$\forall 1 \leq j \leq k : \sum_{p \in p_j} f(p) \geq d_j \tag{III.3}$$

$$\forall q : \sum_{e \in Grid(q)} A_e \cdot \sum_{p : e \in p} f(p) \leq A \tag{III.4}$$

$$\forall p : f(p) \geq 0 \tag{III.5}$$

The objective is to minimize total NoC power consumption, which is the sum of all the communication flows over per-bit power consumption of all the edges (as in Equation (III.1)). In constraint (III.2), we ensure that global latency requirement is satisfied. NoC average latency can be derived by dividing global latency by total communication demands. Constraint (III.3) guarantees that communication demand for each sender/receiver pair is satisfied. Constraint (III.4) states that the total routing channel dimension is limited by area budget A on every grid area $Grid(q)$ of the routing channel.

The following formulation is for pairwise average latency constrained minimum power NoC design. It is very similar to the global average latency constrained formulation, except for constraint (III.7), which formulates that the average latencies of critical commodities are less than the corresponding latency bounds.

$$Min: \quad \sum_{j=1}^{k} \sum_{p \in p_j} \sum_{e \in p} f(p) \cdot P_e \tag{III.6}$$

$$s.t. \quad \forall j \in C : \sum_{p \in p_j} \sum_{e \in p} f(p) \cdot D_e \leq LT_j \tag{III.7}$$

$$\forall 1 \leq j \leq k : \sum_{p \in p_j} f(p) \geq d_j \tag{III.8}$$

$$\forall q : \sum_{e \in Grid(q)} A_e \cdot \sum_{p: e \in p} f(p) \leq A \tag{III.9}$$

$$\forall p : f(p) \geq 0 \tag{III.10}$$

III.3.B Isomorph-Free Exhaustive Topology Generation

The search space of NoC topologies is extremely huge. Even excluding those topologies who are isomorphic, the number of the possible topologies still approaches $O(2^{n^2})$ for n nodes. In our methodology, we restrict the topology search space to *regular topologies*, where each row and column have identical connections. In this way, the number of combinations is significantly reduced, and this regularity makes circuit design and layout easier.

Regular topologies cover a wide range of popular network topologies. We find that any $2n$-dimensional binary hypercube can be mapped to a regular topology, by proving Theorem 4.

Theorem 4 *A $2n$ dimensional binary hypercube can be mapped to a 2^n by 2^n array with* regular topology.

Proof: For a node on the i^{th} column and the j^{th} row of an $n \times n$ array, assign a binary number $(i-1)_2(j-1)_2$ as the id of that node, where $(i-1)_2$ is the binary representation of the number $i-1$. We connect two nodes if and only if their id differs by only one bit. The resulted graph is both a $2n$ dimensional binary hypercube and a *regular topology.*

Most of the popular NoC topologies, such as two-dimensional mesh and torus, high-dimensional mesh and torus, octagon, and twisted cube, etc., can be mapped to regular topologies.

We generate connected topologies on n nodes using *nauty* [38] [1]. We set MAX_DEGREE as an upper bound on node degree. The maximum node degree limits network router input/output ports, since excess number of ports may dramatically increase the network router area and its power consumption.

Table III.2 lists the number of connected topologies on n nodes with different MAX_DEGREE. For a set of 8 nodes, when MAX_DEGREE equals to 4, there are 1929 distinct topologies.

Table III.2: # of Isomorph-Free Topologies

MAX_DEGREE	$n = 4$	$n = 5$	$n = 6$	$n = 7$	$n = 8$
3	6	10	29	64	194
4	6	21	78	353	1929

After we generate all connected isomorph-free topologies on n nodes, we enumerate all linear placements of them. Figure III.4 shows an example of two mappings of a ring structure onto a row of four tiles. Different linear placements lead to different NoC power consumption and delay time.

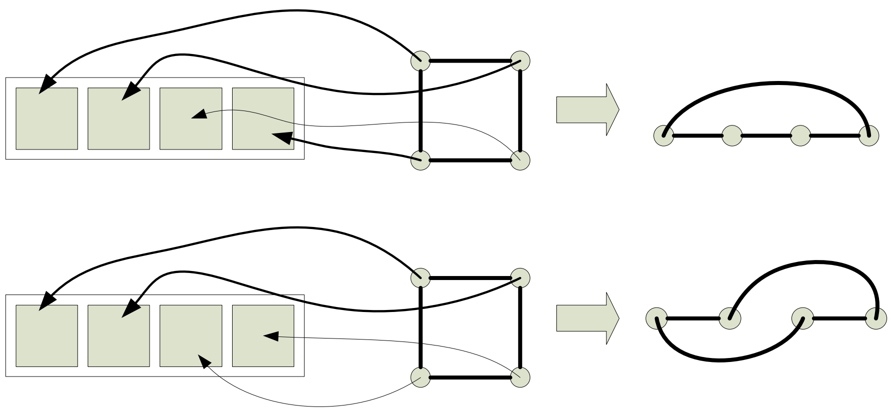


Figure III.4: Example of Linear Placements

Due to symmetry of certain subgraphs, different placements may correspond to the same mapping. We use link bit vector to remove duplicated placements. As shown in Figure III.5, there are six possible links on four nodes, hence a 6-bit link bit vector can represent a placement on four nodes. We then use an array of link bit vectors to keep track of all exist placements and remove those duplicated ones in our mapping algorithm.

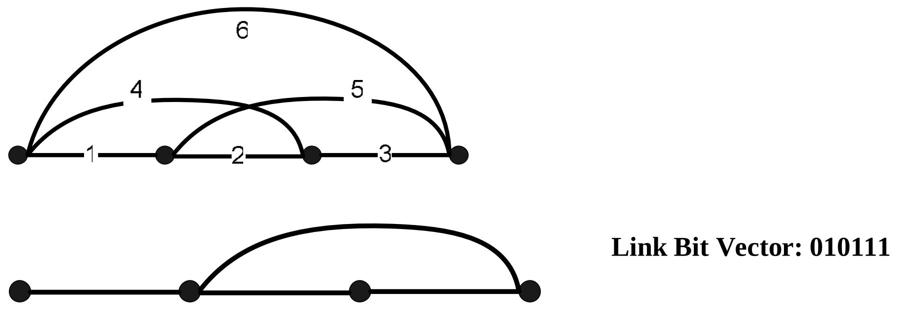


Figure III.5: Link Bit Vector to Represent a Placement

Since total wire length is tightly correlated to NoC power consumption, we only consider the placements with good quality, i.e. with small total wire length. When generating placements, we set an upper bound for total wire length. We map a topology to only those placements whose total wire length is no more than the threshold (as seen in Table III.3) times the minimum wire length of that topology. Our experiments show that this heuristic strategy works fine, for a 4×4 NoC, the placements with minimum wire length always consume minimum power.

Once we generate all the placements on a row/column of on-chip tiles, we can duplicate it to all rows/columns to generate the final NoC topologies. Table III.3 gives the number of final NoC topologies with MAX_DEGREE=3 for one dimensional and maximum node degree of 6 for two dimensional $n \times n$ NoC. Our following power evaluation experiments are all based on these topologies.

Table III.3: # of Regular Topologies on nxn NoC

size	4×4	5×5	6×6	7×7	8×8
threshold	2.0	1.5	1.2	1.1	1.0
# of regular topo	36	254	534	1306	2092

III.3.C Power and Delay Models

Interconnects and network routers are two main contributors to NoC power consumption and communication latency. We adopt the concept of *bit energy* proposed in [55] to represent energy consumption when one bit of data is transported through the interconnects or routers. For each network link e, we assume P_e represents bit energy on link e and the corresponding router, and D_e represents delay on link e and the corresponding router.

$$P_e = P_w + P_r; D_e = D_w + D_r$$

where P_w and P_r are bit energy on interconnects and routers, D_w and D_r are delay of unit flow on interconnects and routers, respectively. When a flow of amount f passes the link and the corresponding router, the power consumption is: $P = P_e \cdot f$, and latency is: $D = D_e \cdot f$.

Interconnect Wires

To achieve high performance low power, many wire technologies have been proposed for on-chip interconnects, such as RC wires with repeated buffers, and on-chip transmission lines, etc. Table III.4 reviews the comparative advantages of three on-chip wire styles.

RC wire with appropriately spaced repeaters is the most common and simplest means of global interconnect [25]. With inserted buffers, its wire delay is improved to linear with total wire length, and the wire bandwidth is increased substantially. However, power consumption of RC wires with repeated buffers increases linearly with the total wire length, hence it is not power-efficient for long distance on-chip interconnects. Increasing the spacing between wires can reduce power consumption, but costs more on-chip area resources.

Transmission line is appropriate for long distance inductance-dominant high-frequency on-chip interconnects. For length of 20mm, transmission line is five times faster and consumes 50% less power than that of RC wires with repeated

Table III.4: Comparison of Different On-Chip Wiring Technologies

Wire styles	**Pros.**	**Cons.**
Buffered *RC* Wire with Minimum Spacing	highest wiring density, most area efficient	highest per-bit energy, longest latency
Buffered *RC* Wire with Large Spacing	shorter latency, lower per-bit energy compared with minimum spaced *RC* wires	more routing area usage than minimum spaced wire
On-Chip Transmission Line	shortest latency, low per-bit energy for long-distance communication	largest routing area, large initial power, largest design effort

buffers [8]. However, due to its complexity at transmitter and receiver circuits, transmission line has a large setup power overhead, hence not suitable for short distance interconnects. Transmission line also requires much wider wire pitch to transfer signals, taking more on-chip area resources.

Since different types of interconnect wire styles have different trade-offs on power consumption, communication latency and area resources, we assume that each on-chip network link can be composed of multiple types of wire styles. We assume four types of wire styles are available for interconnects, namely, *RC* wires with repeated buffers with wire pitch varying from 1×, 2×, and 4× minimum global wire pitch, and on-chip transmission line with $16um$ wire pitch.

For RC wires with repeated buffers, we assume P_w and D_w are proportional to wire length, i.e. $P_w =$ per grid length big energy × wire length and $D_w =$ per grid length delay × wire length. For on-chip transmission line, comparatively large setup costs should be added to D_w and P_w. We use transmission line model proposed by Chen et *al.* [9] to estimate transmission line bit energy and delay.

Table III.5 lists bit energy and delay per grid length (2mm) of these four types of wire styles in $0.18um$ design technology. The supply voltages, wire geometries and device parameters are from ITRS [2]. For RC wires with repeated buffers, the repeaters are inserted to minimize wire delay. Setup costs of 50ps and 4.4pJ/bit are added to D_w and P_w for transmission lines.

Table III.5: Delay Model of Wires

wire type	RC-1x	RC-2x	RC-4x	T-line
P_w (pJ/bit)	2.68	2.15	1.99	0.15
D_w (ns)	0.127	0.112	0.100	0.020

Network Routers

To estimate router bit energy P_r, we use a power simulator *Orion* [53]. We assume 1GHz frequency, 4-flit buffer size, 128-bit flit size. When the number of router input/output ports increases, P_r increases almost linearly. We use the router delay model proposed by Peh et *al.* [43] to estimate NoC router delay.

Table III.6 shows bit energy and latency of routers in 0.18um technology node. When the number of router input/output ports increases, P_r increases almost linearly, and D_r increases in a slower pace.

Table III.6: Power Model of Routers

ports	3	4	5	6	7	8
P_r (pJ/bit)	0.33	0.44	0.55	0.66	0.78	0.90
D_r (ns)	0.662	0.709	0.756	0.788	0.819	0.835

III.4 Experimental Results

Our experiments are on NoC in 0.18um design technology. We assume that grid length are 2mm, and communication demands are evenly distributed, i.e., the bandwidth requirements between every pair of tiles are 1Gb/s. The experiments are based on power/delay parameters described in subsection III.3.C. We use the topology library generated in subsection III.3.B as candidate topologies for design selection. In the MCF approximation algorithms, we set error tolerance ϵ to 1%. In following subsections, we show the impact of wire style optimization, topology selection and tradeoffs between power/latency optimization in NoC design. Since each grid has the same vertical and horizontal dimension, for convenience, we use only the vertical dimension to represent the area budget. This is why the unit of area in our experiments is *um*.

III.4.A Wire Style Optimization

We first demonstrate the power/latency improvement by wire style optimization. Without wire style optimization, we assume only the basic RC wires with repeated buffers available for on-chip interconnects, whose wire pitch is 1× minimum global pitch. With wire style optimization, other three types of wires (see Table III.5) are also available for on-chip interconnects.

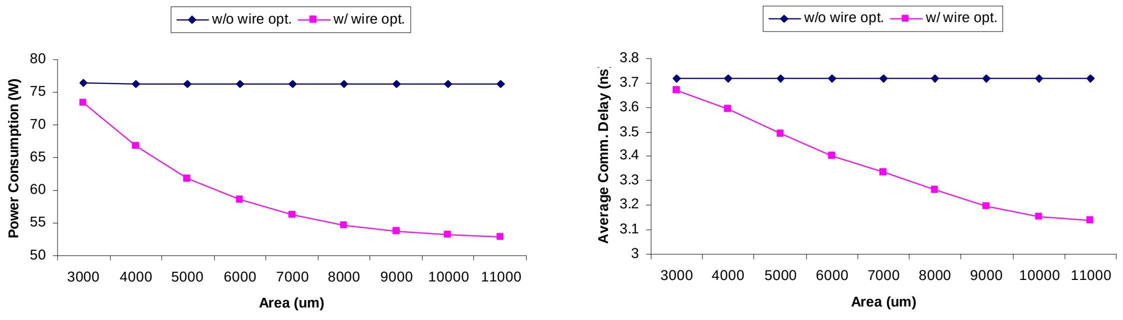


Figure III.6: Impact of Wire Style Optimization

For 8x8 torus networks, Figure III.6 shows its power/latency improvement under various on-chip area resource. When more area is available for wire style optimization, more power/latency savings can be seen. The lower bound of area resource is 3000*um* so that all communication demands be satisfied; when area resource increased to 11000*um*, wire style optimization reaches the maximum impact, up to 30.7% power improvement (from 76.2W decreases to 52.9W) and up to 15.6% latency improvement (from 3.72ns to 3.14ns) can be achieved. When area resource decreases to the bottleneck, 3.8% power saving and 1% latency saving are still seen from wire style optimization. We observe that at flow congested area, minimal pitch wire style is used. However, at uncongested on-chip area, we still can use power efficient wire styles to reduce NoC power consumption and latency.

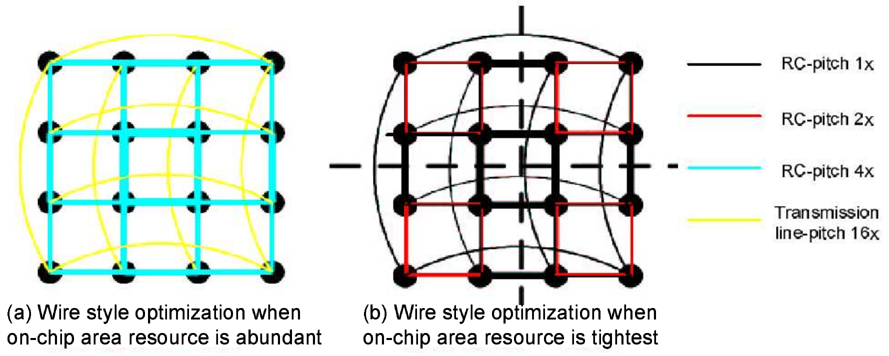

RC-pitch 1x
RC-pitch 2x
RC-pitch 4x
Transmission line-pitch 16x
(a) Wire style optimization when on-chip area resource is abundant
(b) Wire style optimization when on-chip area resource is tightest

Figure III.7: Details of Wire Style Optimization

Figure III.7 shows details of wire style optimization, e.g. the types of wire styles for interconnects and their capacities. The different types of lines represent different wire styles, as shown in the legend. The line width represents the wire capacity. For a 4×4 torus, Figure III.7(a) shows the wire style assignment under loose area constraints. Due to the relatively large available on-chip area resources, transmission lines are selected for long interconnects, and RC wires with repeated buffers with largest spacing (wire pitch is equal to four times minimum pitch) are selected for short connections. Figure III.7(b) shows the wire style assignment under tightest area constraints. Since the communication reaches the maximum capacity, for all interconnects on the two congested cuts shown by dot lines, only the RC wire with minimum pitch is selected because it provides the highest cross-section bandwidth. For those uncongested links, wider wires with lower power consumption are selected.

III.4.B Power Consumption and Latency Tradeoffs

To demonstrate tradeoffs between power consumption and average latency in 8×8 NoC design, we show power savings when a small amount of communication latency is sacrificed. First, we use MCF model to search the topologies with the minimum latency (no power optimization), then loosing this latency constraint by up to 10% and optimize NoC power consumption. Figure III.8 shows the results. The x-axis represents average latency. The y-axis represents power consumption. Each curve represents latency constrained minimum power consumption under certain area budget.

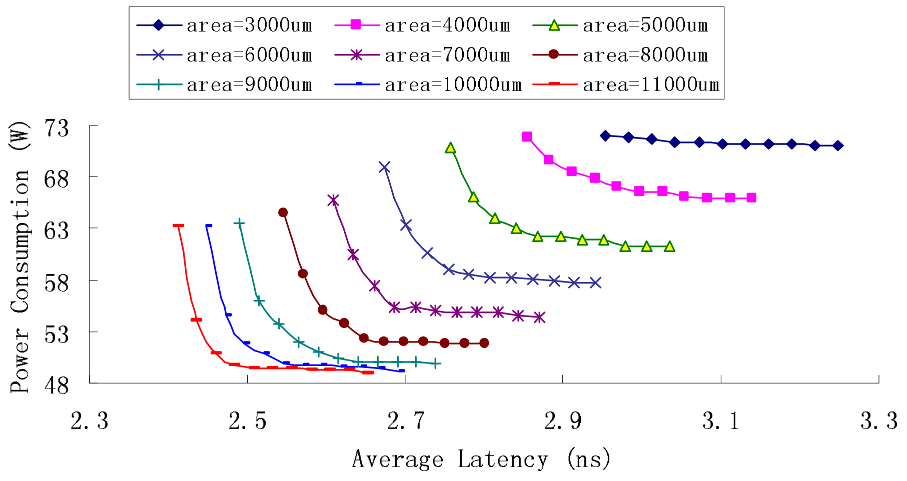


Figure III.8: NoC power and latency tradeoffs

As area budgets increase, the curves move toward left-bottom due to wire style optimization, because those aggressively optimized but area-consuming wire styles, such as transmission lines, can be adopted to optimize both power and latency. When area increases from $3000um$ to $11000um$, minimum latency drops 18.3%, from 2.95ns to 2.41 ns; average power consumption drops 28.3%, from 71.4W to 51.2W.

The slopes of the curves indicate the power consumption reduce rate when communication latency is increased. Take the curve with area $11000um$ as example, when latency constraint is loosened 2%, from 2.41ns to 2.46 ns, the power consumption is reduced from 63.2W to 50.9W, which is a 19.4% improvement. When area is small (3000*um*), the curve is almost flat. This is because area resource becomes bottleneck and flow is congested on chip, so that loosing latency constraint will not bring much benefit.

III.4.C Topology Selection

In this section, we compare these optimal topologies found by our design flow in the topology library with traditional topologies, such as mesh, torus and hypercube. Same as in subsection III.4.B, we set the latency constraints by loosening the minimum latency by up to 10%.

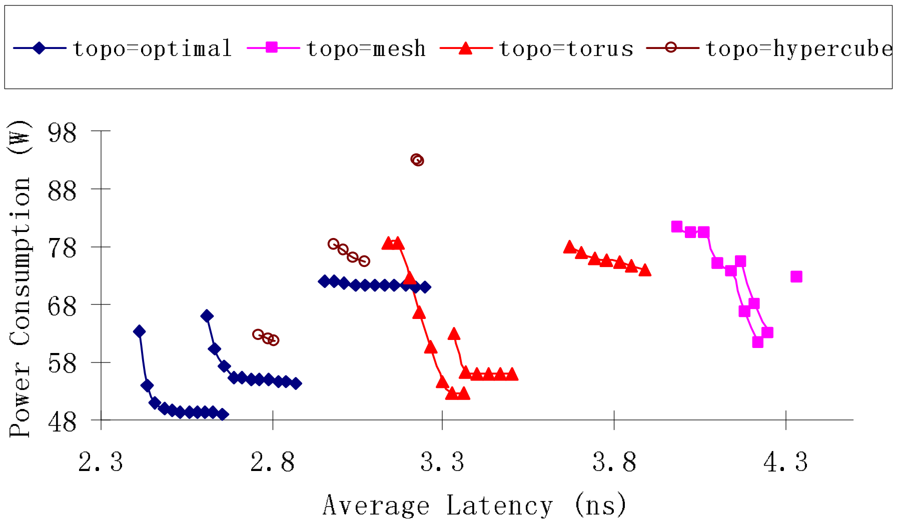


Figure III.9: Power latency tradeoffs among various topologies

Figure III.9 shows comparison among these four types of topologies under different budgets. The x-axis represents average latency. The y-axis represents power consumption. Each group includes 3 curves, 11000*um*, 7000*um*, and 3000*um*, which represents loose, moderate and tight area constraints, respectively.

For a certain topology, since larger latency constraints lead to smaller power consumption, and vice versa, we pick the point with minimum power latency product for a quantitative comparison, as shown in Table III.7.

Table III.7: Topology Comparison

area (um)	**topo**	**L (ns)**	**P (W)**	**P*L (W*ns)**	**Impv. (%)**
3000	mesh	4.34	72.7	315.2	**26.7**
	torus	3.74	76.1	284.7	**18.9**
	cube	3.23	92.8	299.8	**23.0**
	optimal	3.25	71.1	**230.9**	
7000	mesh	4.25	63.0	267.9	**44.5**
	torus	3.37	56.3	189.6	**21.5**
	cube	3.04	76.0	231.2	**35.6**
	optimal	2.69	55.4	**148.8**	
11000	mesh	4.22	61.2	258.3	**52.1**
	torus	3.33	52.7	175.3	**29.4**
	cube	2.76	62.6	173.1	**28.5**
	optimal	2.48	49.8	**123.8**	

The first two columns list area budgets and topologies. Column 3-5 show latency, power consumption, and power latency product for certain topology under given area budgets. The sixth column of the table lists the improvement in terms of power latency production, when compare our selected optimal topology with mesh, torus and hypercube.

From the table, we observe that mesh is not a desirable topology for NoC of size 8x8. Compared with other topologies, its latency is quite large, because data packets need many hops to arrive the destinations. Also it lacks of long global links and doesn't make fully use of wire style optimization, so that when area budgets increase, its power consumption is not as good as torus. Torus and hypercube have their own advantages. In general, torus is better in terms of power consumption, since it has simpler network router architecture; hypercube is better in terms of latency, since it has a lot of shortcut links.

Our selected optimal topologies show big advantages over the other three traditional topologies. They have small power consumption and latency. In terms of power latency production, they achieve an improvement up to 52.1% compared to mesh and 29.4% compared to torus (area = $11000um$), and 28.5% compared to hypercube (area = $7000um$). Figure III.10 shows the connections on one raw of our

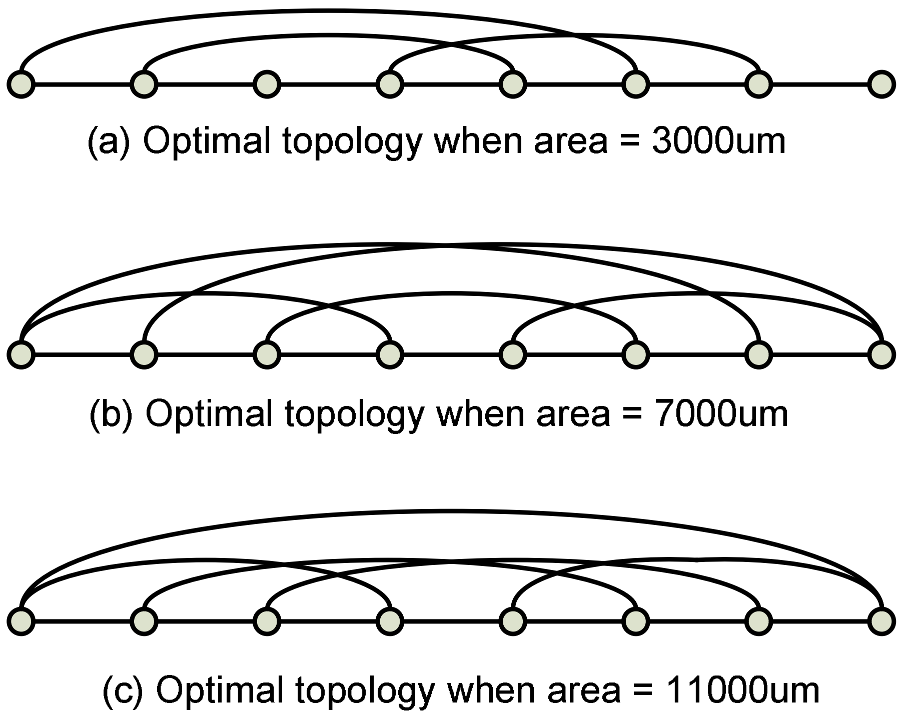

(a) Optimal topology when area = 3000um
(b) Optimal topology when area = 7000um
(c) Optimal topology when area = 11000um

Figure III.10: Optimal topologies under various areas

selected optimal topologies under each area budget. Duplicating these connections to every row and column will generate the final topology design.

III.4.D Pairwise Latency Constraints

We also conduct the experiments which include pairwise latency constraints. Thus instead of giving a global latency constraint, we set latency constraint for each communication between a pair of nodes. In practice, the communications between a pair of nodes which are close usually have a low latency since there are less number of hops as well as shorter wires between them; however, the communications which separated further tend to have larger latencies therefore become the "critical commodities". Consequently, in our experiments on 8×8 NoCs, we regard those commodities whose separation distance is 7 units of grid length or further as critical commodities. Also, we use another parameter called "latency constraint coefficient" γ to set the latency for each specific critical commodity: first we calculate a "base latency" for each commodity. The base latency is computed by assuming 1x RC wire is used, the distance is Manhattan, and routers have minimum degrees. Then we set the latency as $\gamma \times base\_latency$. Note that γ could be less than 1 as RC wires with larger pitches or transmission lines could achieve faster speed.

Figure III.11 shows the power and latency tradeoffs with pairwise constraints when the routing area is 3000 *um*, 5000 *um* 6000 *um* and 7000 *um* respectively. The x-axis represents the latency constraint coefficient γ and the y-axis represents power consumption. It is observed that γ could be reduced as small as 0.8 except for the tightest area constraint 3000 *um*. When γ is 0.8, all the three curves converge to one point, which indicates that the topologies with short cuts among the critical commodities must be used with the tightest constraint; however, when γ becomes larger, more alternative topologies could be chosen, therefore larger routing area could achieve smaller power consumption.

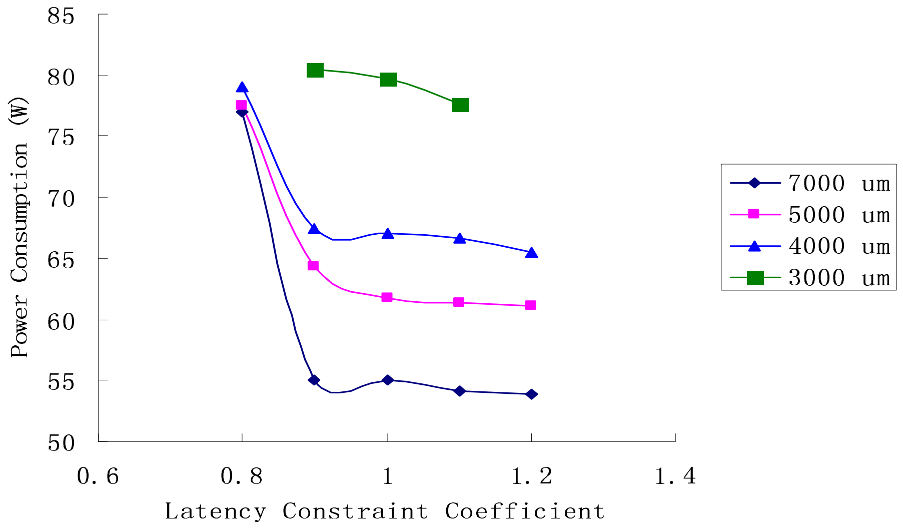


Figure III.11: NoC Power and latency tradeoffs with pairwise latency constraints

III.4.E MCF Performance Improvement

To demonstrate the efficiency of the proposed algorithm, we conduct experiments to compare its CPU time with the linear programming (LP) solution produced by CPLEX, a commercial LP solver. We choose torus as the representative topology to make the comparison. We test the performance on 3000, 7000 and 11000 as small, moderate and large grid area, by scaling down them by the factor of $k^4/8^4$ for the $k \times k$ case, by approximating the communication demands to be k^4. All the experiments are conducted in a PC with 2.8 GHz CPU and 784MB memory, and CPLEX 9.1 is used. The detail results are shown in Table III.8, where columns 3–6 show the result values and CPU time (in seconds) of CPLEX and our approximation algorithm respectively, column 7 shows the gap between the approximate results and the optimal solutions, $(col5 - col3)/col3$, and column 8 shows our speedup, $col4/col6$.

The table shows that our proposed algorithm can obtain correct results within the 1% threshold, which is our input settings. Also, it is much faster than the LP solver, and becomes more and more significant when the size becomes larger: in the 7×7 cases, it has been more than 100 times faster than CPLEX. In 8×8 cases, the approximation method also runs fast, while CPLEX is too slow to produce any results.

III.5 Summary

We study the tradeoffs between NoC power efficiency and average latency. By adopting an MCF formulation, we are able to reduce power consumption of NoC under given latency constraint, through simultaneous optimization of network topologies and wire styles. Experimental results suggest that for NoC of size 8×8 (1) Power and latency co-optimization is critical in NoC design. With 2% latency overhead, up to 19.4% power savings can be seen. (2) compared with mesh, torus and hypercube topologies, our optimized design can improve power latency product

Table III.8: MCF Performance Improvement

Size	**Area**	**CPLEX**		**Approx.**		**Err**	**Speedup**
		Obj	**CPU**	**Obj**	**CPU**	**(%)**	
5×5	473	6611	105	6652	11	0.62	9.55
	1069	5389	104	5430	11	0.76	9.45
	1679	5193	10	5234	12	0.78	0.83
6×6	950	16830	1496	16955	65	0.74	23.02
	2215	13195	1910	13298	29	0.78	65.86
	3481	12580	291	12683	29	0.82	10.03
7×7	1759	36860	9963	37156	78	0.80	127.73
	4104	28405	15040	28641	46	0.83	325.96
	6488	27464	8280	27689	56	0.82	147.86
8×8	3000	N/A	N/A	73315	113	N/A	N/A
	7000	N/A	N/A	56207	48	N/A	N/A
	11000	N/A	N/A	52915	62	N/A	N/A

by up to 52.1%, 29.4% and 35.6%, respectively.

Chapter III includes the contents of two published papers. "Communication Latency Aware Low Power NoC Synthesis," by Y. Hu, Y. Zhu, H. Chen, R. Graham, C.K. Cheng, in Proceedings of 43rd ACM/IEEE Design Automation Conference. "Physical Synthesis of Energy-Efficient NoCs Through Topology Exploration and Wire Style Optimization," by Y. Hu, H. Chen, Y. Zhu, A. A. Chien, C.K. Cheng, in Proceedings of 23th IEEE International Conference of Computer Design. The dissertation author was the researcher and co-author of both papers.

IV

Supercomputer Interconnection Networks Synthesis and Optimization

IV.1 Overview

In this chapter, we propose a design methodology that is able to select the best interconnection network topology among a large number of candidates so that the average communication latency is minimized. The major contributions of our work are as follows:

1. We propose a fully automated design flow that is able to evaluate thousands of network topologies and find the best candidate according to the available technology, physical constraints and applications. All these conditions, such as unit wire latency, router latency, board dimensions, pin numbers, or communication patterns, are the input and parameters of the flow, and thus can be specified and modified by users. This feature enables our methodology to be applicable for different supercomputer systems, with little modification.

2. We demonstrate our flow by using the the packaging framework of Blue

Gene/L supercomputer. We conduct the experiments using different input parameters. The optimal designs we find in a midplane under different circumstance can improve 12% – 56% of the average communication latency compared to the original 3D torus design, which shows the effectiveness of our methodology.

3. We develop a topology generation scheme which is able to cover a large design space without loss of the regularity. Here, it is applied to the networks on the two-level graph model of the 512 processors and able to produce more than 2000 representative topologies, offering great freedom for designers. Furthermore, The link capacity (the number of wires needed in each link) can also be determined accordingly.

4. The current design flow is also easy to extend to minimize the power consumption, or perform the latency-power co-optimization, with appropriate power models for routers and wires. The power consumption is also an important concern for supercomputers. Knowing the tradeoffs between the latency and power enables designers to make correct decisions to build supercomputers.

The rest of this chapter is organized as follows. Section IV.2 will introduce our design flow and the general formulation to evaluate the performance of interconnection networks. In Section IV.3, we will use the packaging framework of Blue Gene/L as a concrete example to demonstrate our design flow; the generalized models and parameters described in Section IV.2, such as wire and router delay models, and physical constraints, will be specified using practical data. The experimental results will be presented in Section IV.4. Summary will be given in the last section.

IV.2 General Design Flow & Formulation

Figure IV.1 shows the general design flow of our methodology. Users need to provide four categories of inputs to perform the topology synthesis:

1. **Topology Pool**: The topology pool contains all candidate topologies users want to use. They can be created by the users or generated by other tools, either regular or irregular. The pool can contain as many candidates as designers wish since our evaluation algorithms are very efficient (in the experiments we evaluate more than 2000 topologies). The best topology will be selected based on our evaluation.

2. **Delay Models**: Users need to specify the wire and router delay models, based on the wires they use and the router architectures. They are used to calculate the communication latency, which is our objective.

3. **Communication Patterns**: The communication patterns among processors will be fed into our evaluation algorithm so that the synthesized network is able to satisfy the communication demands.

4. **Physical Constraints**: Users also need to discover the physical constraints which must not be violated, including the board dimensions, number of layers, number of pins, and number of connectors.

All these inputs are fed into the MCF solver to evaluate the performance and the best topology will be selected according to the communication latency. The number of wires on each link will also be determined by the solver, with all the communication demands and physical constraints satisfied.

The entire flow is automated. The MCF solver is able to take all the inputs, perform the evaluation, and provide the synthesis results which can be used by the designers. As the core component of our flow, we introduce the formulation of the MCF evaluation solver.

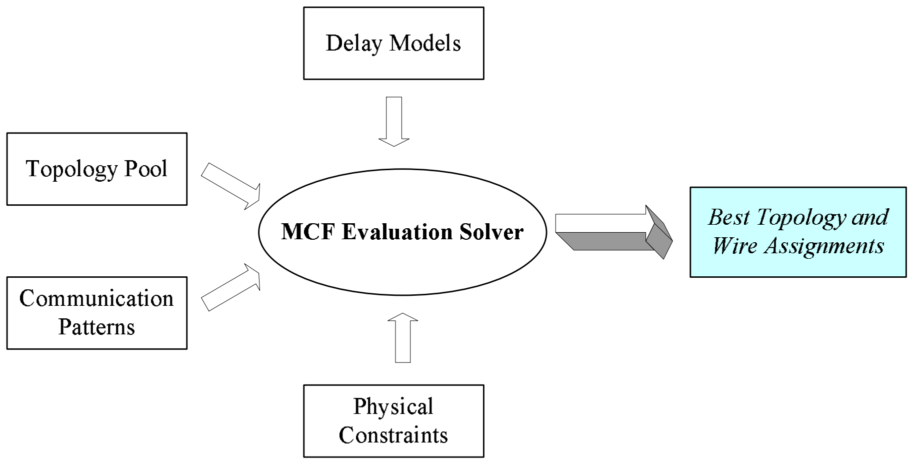


Figure IV.1: Design Flow

We model a given interconnection network topology as a graph $G(V, E)$, where $|V| = n$, $|E| = m$. Each node represents a computing module (processor) and each edge (a, b) represents a link between processors a and b. A set of k communication demands are given, i.e. d_j represents the demand from processors (nodes) s_j to t_j, $\forall 1 \leq j \leq k$. The communication demands represent the demand for bandwidth between each pair of nodes; this quantity becomes the injection rate when averaged over the lifetime of the network that we profiled. Our task is to determine how many wires need to be assigned for each link, so that the physical constraints are satisfied and the overall average latency is minimized.

The above statement can be formulated as the MCF problem [5] if we consider the communication demand from s to t as a commodity with source node s to sink node t, and the number of wires for link $e \in E$ as the flow $f(e)$. The average communication latency consists of two portions: the router delay and the wire delay. If we assume there is a router with each node, our objective can be written as follows:

$$\sum_{e \in E} f(e) \cdot D^R_{g(e)} + \sum_{e \in E} f(e) \cdot D^W_e \tag{IV.1}$$

where $D^R_{g(e)}$ represents the delay for the router in node $g(e)$ that is the starting node of edge e, and D^W_e represents the delay for link e. Therefore the first part of the objective denotes the total latency on routers and the second part is the latency on wires.

The primary constraint is that communication demands must be satisfied. Let $f^j(e)$ denote the amount of flow dedicated for communication j $(1 \leq j \leq k)$ from node s_j to node t_j, then the following constraints must be satisfied:

$$\sum_{j=1}^{k} f^j(e) = f(e), \quad \forall e \in E \tag{IV.2}$$

$$\sum_{e \in E_i^{in}} f^j(e) - \sum_{e \in E_i^{out}} f^j(e) = -d_j, \quad i = s_j, 1 \leq j \leq k \tag{IV.3}$$

$$\sum_{e \in E_i^{in}} f^j(e) - \sum_{e \in E_i^{out}} f^j(e) = d_j, \quad i = t_j, 1 \leq j \leq k \tag{IV.4}$$

$$\sum_{e \in E_i^{in}} f^j(e) - \sum_{e \in E_i^{out}} f^j(e) = 0, \quad i \in V - \{s_j, t_j\}, 1 \leq j \leq k \tag{IV.5}$$

where E_i^{in} represents the set of edges that point to node i and E_i^{out} represents the set of edges that start from node i, and s_j and t_j are the source and sink nodes of the commodity j, respectively. Constraint (IV.2) means the flow on an edge is the sum of the flow of all commodities crossing over that edge; constraints (IV.3) – (IV.5) indicate the difference of flow amount that arrives and leaves node i, for commodity j: for the source node s_j, it is $-d_j$; for the sink node t_j, it is d_j; and the difference is 0 for the rest of the nodes. These are the conventional flow conservation constraints for MCF [5].

The physical constraints restrict the number of wires we are able to use. In our general formulation, we consider two types of physical constraints:

- Intra-board constraints: a set of wires that link the nodes in different regions in one board must not exceed the cross section of the board. The resource used for each wire can be estimated by the wire pitches and the available cross section can be calculated by the board dimension and the number of signal layers:

$$\sum_{e \in E_q^{cross}} f(e) \cdot pitch \leq W_q \cdot L_q \tag{IV.6}$$

where E_q^{cross} is the set of edges that pass over the cross section q, W_q is the available cross section width and L_q is the number of signal layers in the board.

- Inter-board constraints: the number of wires connecting nodes on different boards are usually constrained by the number of pins in the board connectors. Similarly, we can write the following constraint:

  $$\sum_{e \in E_r^{pin}} f(e) \cdot K_r^{pin} \leq C_r \cdot V_r^{pin} \tag{IV.7}$$

  where E_r^{pin} is the set of edges that cross over the connector r, K_r^{pin} is the number of pins needed for each wire, C_r is the number of available connectors, and V_r^{pin} is the number of pins in each connector.

The above is a very general formulation to evaluate the communication latency in a supercomputer system with the given topology. It is applicable to any multiprocessor interconnection network by deriving the delay models and setting the parameters accordingly. In Section IV.3, we will give a concrete example using Blue Gene/L supercomputer, by presenting the delay models and physical constraints.

IV.3 Topology Synthesis in Blue Gene/L: An Example

In this section, we will use our design flow to perform the topology synthesis on the interconnection network in one midplane of the Blue Gene/L supercomputer. We shall first give an overview on the system, then describe the details of the important components in our design flow.

IV.3.A Overview

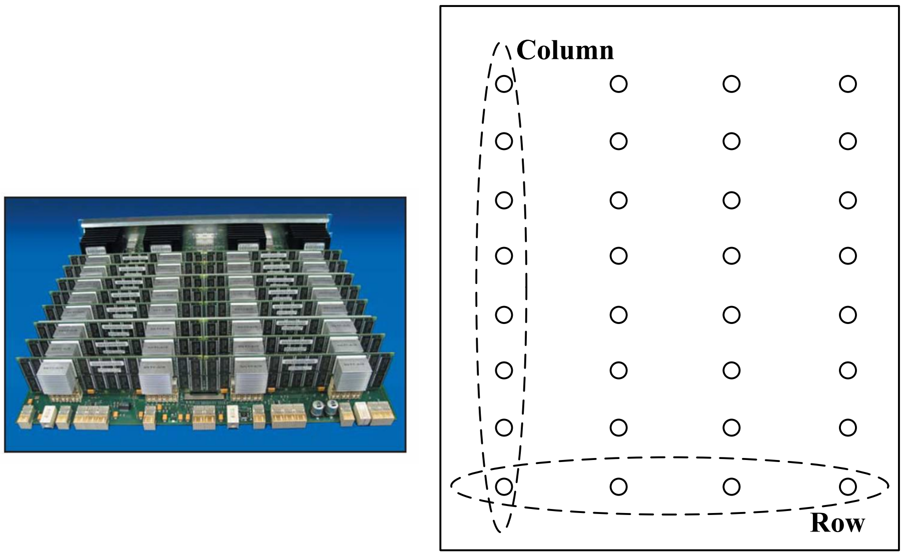
Blue Gene/L computer is a massively parallel supercomputer based on IBM system-on-chip technology. It is designed to scale to 65, 536 dual-processor nodes (computer ASICs) [7] [16]. The entire system is organized as 2 nodes per compute card, 16 compute cards per node card, 16 node cards per 512-node midplane, 2 midplanes in a 1024-node rack, and totally 64 racks. The nodes are connected using 3-dimensional torus networks. Each 32-node node card contains a $4 \times 4 \times 2$ torus and each 512-node midplane contains an $8 \times 8 \times 8$ torus. Link ASICs are used to construct the networks connecting different midplanes [19]. In this work, we will consider the networks within one midplane, i.e. the topologies that connect 512 nodes.

In our work, we make the following assumptions:

- We follow the same hierarchical structure of midplane/node card/compute card in our design. The number of nodes in each board remains the same too.
- The properties of the boards, including dimensions, number of layers and dielectric keep unchanged;
- We will seek better topologies than the existing 3D torus to implement the networks in the Blue Gene/L midplane.

IV.3.B Graph Models & Topology Generation

We use a 512-node graph to model a midplane, where each node in the graph represents a computer ASIC. Since the ASICs in a midplane are organized hierarchically, we also build a two-level graph model that captures the topologies in one node card (low-level) and the entire midplane (high-level) respectively. The low-level graph therefore consists of 32 nodes, as one node card contains 16 compute cards and one compute card has 2 nodes. The 32 nodes are arranged in 8×4 grids, according to their locations in a node card, as shown in Figure IV.2 (a) [19].



(a) Blue Gene/L Node Card (b) Graph Model in Node Card

Figure IV.2: Blue Gene/L Node Card and Topology

Therefore, we also arrange our low-level graph as 8×4 grids when performing the topology generation. We make the following two assumptions. First, taking the regularity into consideration, we assume that all the rows have the same topology, as well as columns. Second, because the 4 nodes in a row are composed by 2 compute cards, they usually have more communications. Also, since there are only 4 nodes spreading in the horizontal direction, there should be ample space to route the wires. Consequently, we assume these 4 nodes are strongly connected. For the topology in a column, we follow the method in [28] to generate all topologies of 8 nodes, with the limitation that the degree of each node is no more than 3. There are totally 192 isomorph-free topologies. Then we place each topology in a line, which results 2092 different linear placements. They are duplicated to each column. Figure IV.2 (b) shows the organization of the low-level graph. We highlight one row and one column, where we could plug in different topologies and duplicate them to all rows and columns.

In the higher level, there are 16 node cards plugged in the midplane. They are arranged in 8 rows vertically, where each row contains 2 node cards. If

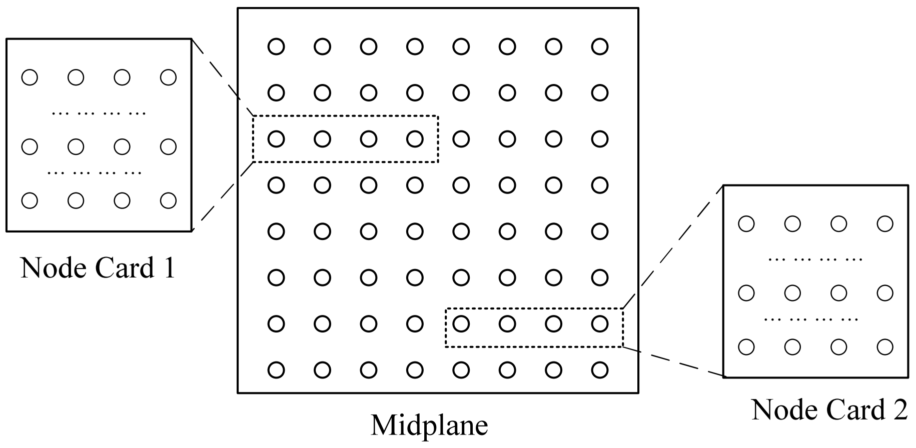


Figure IV.3: Blue Gene/L Midplane and Topology

Table IV.1: Router Delay

Radix	2	3	4	5	6	7	8
Delay (ns)	1.1008	1.3318	1.4958	1.6229	1.7268	1.8147	1.8908

two nodes in different node cards want to communicate, they must go through the midplane.

Thus, in the midplane there will be 64 nodes (4 for each node card), arranged in 8 rows and 8 columns. To generate the topologies in this graph, we still follow the method in [28] to obtain the topology for 8 nodes and duplicate it for both the row and column to impose the regularity.

IV.3.C Delay Models

We should derive the delay models for both wires and routers. We assume that differential wires are used in the Blue Gene/L boards. We estimate the unit length delay as the speed of light in the card dielectrics FR4. Therefore the unit length delay is roughly 7.1 ns/m.

The router delay is related to the radix of routers. We make use of the formula derived in [44] which is based on the logical effort, and assume $90nm$ design technology is used. The delay of routers with different radixes are shown in Table 1.

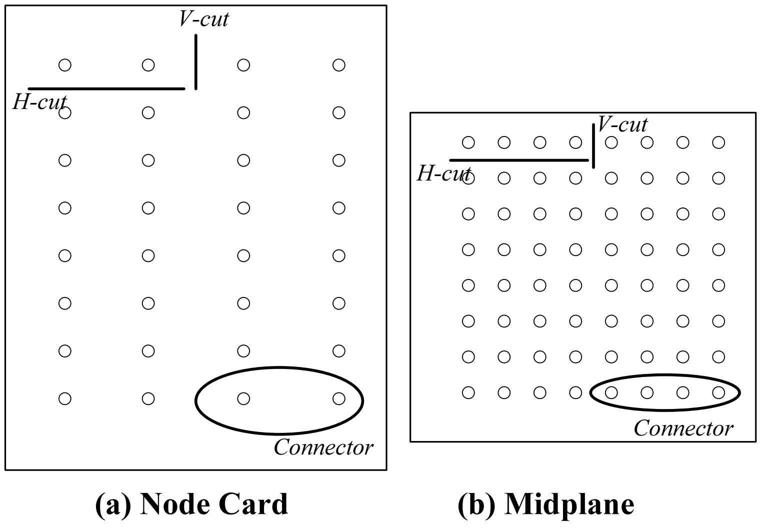


Figure IV.4: Physical Constraints on Node Card and Midplane

IV.3.D Physical Constraints

We should consider two types physical constraints: the available routing area for wires in cards, and the number of pins in the connectors among cards. The first factor is determined by the board dimensions, number of layers and wire pitches, and the second one is determined by the number of connectors and the number of pins per connector, as discussed in Section IV.2. In a compute card, there are only two nodes and there is ample space for routing, so we focus on the constraints on node card and midplane. We demonstrate how the physical constraints are calculated in a node card as follows:

- In a node card, the wires are usually short and the width is $190um$ to $215um$ [19] (in the experiments we choose $200um$). The pitch is set as twice as the width, thus is $400um$. The board dimension is $0.46m \times 0.61m$ [19], where the 4×8 grid of nodes are located. Therefore we estimate the horizontal distance between two nodes is 0.1 m and the vertical distance is 0.07 m. A node card has 6 signal layers [19], 3 are used to route horizontal wires and 3 are used for vertical wires. Hence the horizontal wire capacity, which is constrained by the vertical width, in a row is 0.07*3 = 0.21 (as shown in

"V-cut") in Figure IV.4 (a). In the vertical direction, we should consider two nodes together since they are from the same compute card, as shown in "H-cut" in Figure IV.4 (a), so that will be 0.1*2*3 = 0.6.

- There are 6 FCI Metral 4000 connectors between a compute card and node card, each of which consists of 36 pins [19]. So the total number of wires connected from/to a compute card is constrained by the total pins of these 6 connectors, as shown in the right bottom part in Figure IV.4 (b). We assume half of the pins are used for power ground, and 20% are used for other networks, so the total available pins are 6*36*(1-0.5-0.2) = 64. Because differential pairs are used for wires, there will be 32 wires in total that can be escaped out.

We make the similar estimation on the midplane, as shown in Figure IV.4 (b). Note that when we consider the "H-cut" and connector constraints, 4 nodes are grouped together since they are the 4 virtual nodes for one node card and therefore share the same capacity.

The above estimated parameters are just used in our experiments. In practice, designers can easily change the numbers if more information is available, and therefore obtain customized solutions using their settings.

IV.4 Experimental Results

We implemented the MCF solver using the C language to test the design flow we proposed in a Linux machine with a 2.8GHz CPU and 2GB memory. We generated several sets of test cases with different characteristics to demonstrate the strengths of our methodology and also show the tradeoffs among the constraints and objective. The experiments were conducted on two groups of test cases: the instances in the first group were randomly generated with different characteristics to display the tradeoffs among communication latency, throughput, distribution and physical resources; the instances in the second group were extracted from the

NPB parallel benchmarks to show the strengths of our design flow over practical instances.

IV.4.A Experiments on Randomly Generated Instances

We generate test cases which are the communication pairs among processors. The communication patterns are randomly generated; however, they are controlled by the following parameters:

- *Total number of communication demands* (T): Not all pairs of nodes need to communicate, if proper processor assignments are performed. It is reasonable to assume there are only $O(n)$ pairs of communications, as the examples in the benchmark suites in [17].

- *Communication amount/coefficient* (d): Without loss of generality, we assume all the communications have uniform traffic. However, the traffic amount may vary, which will affect the congestion level of the links and average latency.

- *Communication distribution probability*: During the processor task assignment, we know that the tasks which need to communicate with each other will be assigned to processors that are close together [17]. Consequently, the nodes within one compute board tend to have more communications than the nodes outside the board, and the nodes within one node card would have more communications than the ones located in different node cards. We use two parameters p_1 and p_2 to control the communication distributions: p_1 is the probability that communications happen within one compute card: p_2 is the probability that communications happen within one node card (but different compute cards); and thus $1 - p_1 - p_2$ is the probability that communications happen across the node cards.

We conducted the experiments with three categories of test cases. First we fix the communication distribution probability and study the tradeoffs between the latency and communication throughput; then we adjust the probability of communication distribution and show the change of average latency; finally we want to relax the physical constraints to see how much more we can improve in the future. Meanwhile, we will compare the optimal topologies we found with the 3D torus topology which is currently used by Blue Gene/L.

Latency and Throughput Tradeoffs

To demonstrate the tradeoff between latency and communication throughput, we first fix the communication distribution as $p_1 = 0.4$ and $p_2 = 0.5$; i.e. 40% of the communications happen within a compute card, 50% of the communications happen crossing compute cards but within a node card, and the rest 10% happen across node cards. We use three groups of cases where the total number of communication demands $T = 2048, 3072$ and 4096 respectively (4x, 6x and 8x of the total number of nodes). The latency-throughput tradeoff curves with optimal topology selection are shown in Figure IV.5. The X-axis is the traffic demand coefficient (throughput) for each communication demand (all are uniform); the Y-axis is the average latency, which is the total latency on wires and routers divided by the total amount of communications. Different points indicate that different topologies are used. We also compare the latency values of 3D torus networks and our optimal topologies. The results are shown in Table 2, where "Capacity" means the maximum number of demand amount it can accommodate; "Min Latency" and "Max Latency" denote the average latency when the demand has minimum and maximum values. For 3D torus topology, these two values are the same since there are no topology optimizations.

By analyzing the results, we have the following observations:

- *(i)* Given fixed number of communication demands and patterns, different topologies will be selected with different communication amounts, after the links are congested. We can see the curve is a straight line with low traffic but becomes super linear with the growth of traffic amount. We always prefer a

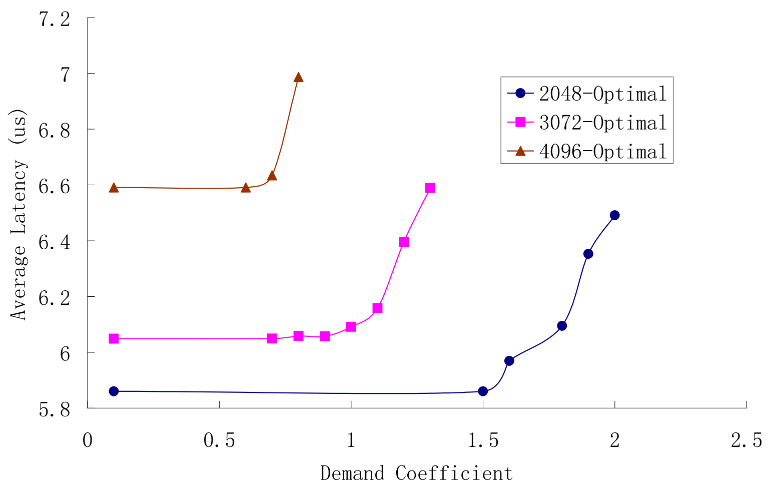


Figure IV.5: Latency-Throughput Tradeoff Curves with Fixed Communication Distribution

topology with less number of hops and also less detours, since such a topology will incur less router and wire delays. When the traffic increases, however, this topology is no longer feasible since it will cause congestion in some nodes/links. Therefore those topologies with more even link distributions will be chosen. Figure IV.6 shows the optimal column topologies in a node card with communication coefficients of 1.5 and 1.9 respectively, when there are 2048 communications. Notice the cut shown between node 4 and node 5: the topology in the above has 3 links across it, while the one below has 4 links. When traffic is less, three links are enough to utilize the flows across the cut; with more traffic, node 3 must be used to share the burden, which will result in more uniform traffic, but sacrifice the local link between node 1 and node 3, and end up with 8.36% more average latency.

- *(ii)* The commonly used 3D torus topology has two weaknesses. First, it cannot accommodate large communication traffic. For example, the maximum coefficient is 1.1 with 2048 communications in a 3D torus network while the optimal topology we find can accommodate a coefficient as large as 2. Second, with the same communication coefficient, the latency in a 3D torus

Table IV.2: Comparison of Optimal and 3D Torus Topologies

#Comm	**Top**	**Capacity**	**Min Lat (us)**	**Max Lat (us)**
2048	Optimal	2	5.86	6.49
	3D Torus	1.1	6.63	6.63
3972	Optimal	1.3	6.05	6.56
	3D Torus	0.7	10.54	10.54
4096	Optimal	0.8	6.59	6.99
	3D Torus	0.3	15.06	15.06

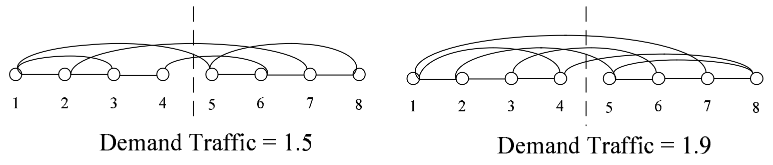


Figure IV.6: Optimal Topologies with Different Parameters

is worse than the optimal topology, especially when there are many communications. As we can see, with 4072 communications, the average latency in a 3D torus network is more than twice as large as that in the network with the optimal topology. This is because the torus does not have enough long links, which results in more hops and detours. Thus, it is of importance to look for better alternative topologies.

- *(iii)* Comparing the latency values for different number of communications with the same communication demand, we find the difference is not much if we use the optimal topologies found by our design flow. This is because we are able to select different topologies and assign flows in order to fully utilize the resources. In contrast, in the 3D torus network, there are large differences among the latency values for different number of communications, which indicates that a fixed topology is easy to become congested and therefore negatively impact the performance.

Latency and Communication Distribution Relations

We study the relations between latency and communication distribution by fixing the number of communications to be 2048, and adjust the fractions of total distributions to generate 3 groups of test cases, with the probability 40%/50%/10%, 40%/40%/20% and 40%/30%/30% respectively, i.e. the fraction of communications that are within a compute card is fixed to be 40%, those exist within a node card vary from 50% to 30%, and 10% – 30% of them happen across different node cards. We also adjust the demand traffic amount to plot the latency-throughput curves. In order to see the trend in different levels clearly, the latency values in low-level (node card) and high-level (midplane) are plotted in separate curves, as shown in Figure IV.7 and IV.8.

Comparing the curves in the two figures, we can see that the 40%/50%/10% curve in the low-level and 40%/30%/30% curve in the high-level demonstrate the super linear property. As mentioned in the previous section, this property implies that different topologies are selected due to the congestion of communication traffic. Therefore this phenomenon indicates that in the 40%/50%/10% cases, the bottleneck occurs in the low-level, while in the 40%/30%/30% cases, the traffic in the high-level is the bottleneck. For the 40%/40%/20% cases, both curves are quite flat, which indicates that they have quite balanced loads in the low level and high level. Hence, following this methodology, besides finding the optimal topologies and flow distributions, we can also identify the bottleneck of the traffic, and therefore adjust the resources accordingly in order to further improve the performance.

Physical Constraints Impacts

In all the above experiments we define the physical constraints by using the board dimensions, connectors and pin numbers used in Blue Gene/L packaging [19]. According to our analysis, the flows are saturated due to the limited number of pins in the connectors between compute cards and node cards, and between

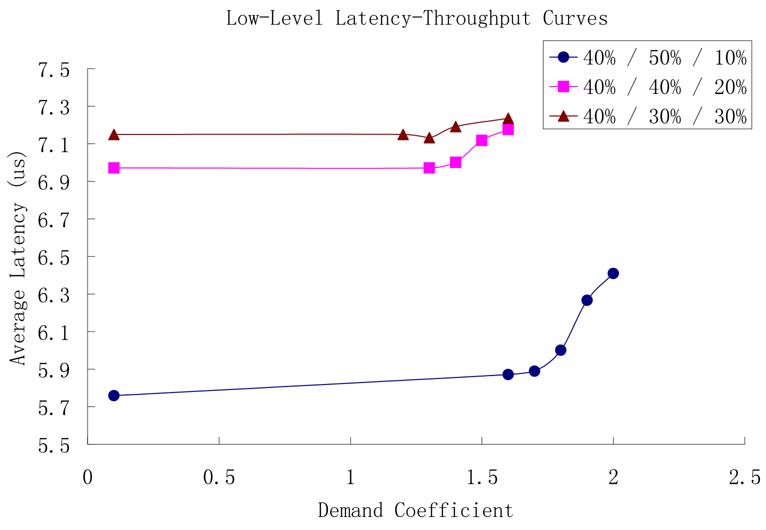


Figure IV.7: Low-Level Latency-Throughput Tradeoff Curves with Different Communication Distribution

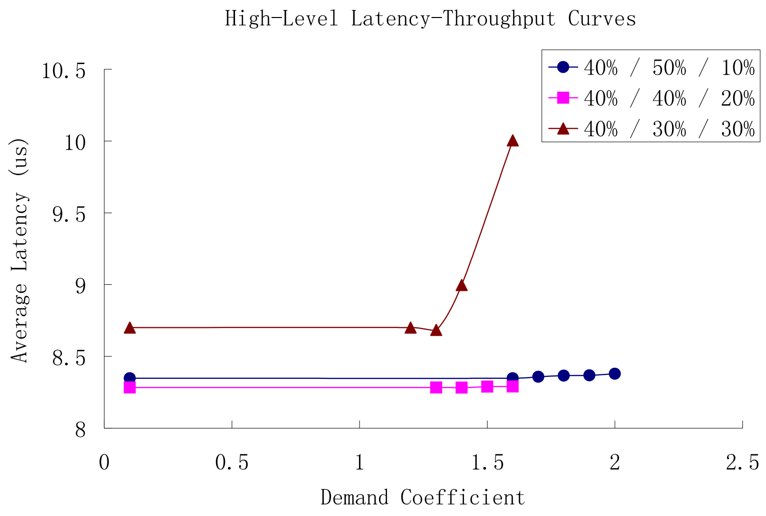


Figure IV.8: High-Level Latency-Throughput Tradeoff Curves with Different Communication Distribution

node cards and midplane. In this part of the experiment, we would like to relax this physical constraint and assume we have more pins available for routing.

We choose a case which has 2048 communications, with the distribution 40%/30%/30%, and demand coefficient 1.6. The previous experiment reports that the average communication latency in midplane is 10.00 with 200 pins (15 connectors) between node cards and midplane. We gradually increase the number of pins and compute the average latency values. The results are shown in Figure IV.9. We find that we can reduce the latency by 14% when the pin number is increased to 320; particularly, latency is reduced by 8% if we only add 20 pins, and 13% with only 40 additional pins. This indicates that when the networks are congested, latency can be greatly improved by adding relatively small amount of routing resources. In fact, when the number of pins is small, we have fewer choices for topologies; when it is increased, the previous optimal topologies become sub-optimal in terms of communication latency. The corresponding topologies for the three points in Figure IV.9 are shown in Figure IV.10: when pins are limited, long links are essential to reduce the intermediate traffic; with the increase of pin number, the long links are no longer preferred since they will increase the degree of routers therefore add the routing complexity. Hence, using this approach, we are able to know which resources are needed in order to improve the latency, and their marginal impacts on the performance.

IV.4.B Experiments on Benchmark Instances

We also demonstrate the strengths of our design flow using the NAS parallel benchmarks [6]. We run the class B benchmarks with 121 processes (for BT and SP) or 128 processes (for other six benchmarks). The traffic patterns are then extracted using Intel Trace Analyzer and Collector 7.1 in Linux platform. Among the eight benchmarks, we exclude EP, FT and IS in our experiments, because there are too few communications among the processes. Table IV.3 lists the characteristics of the rest five benchmarks which are used in the experiments.

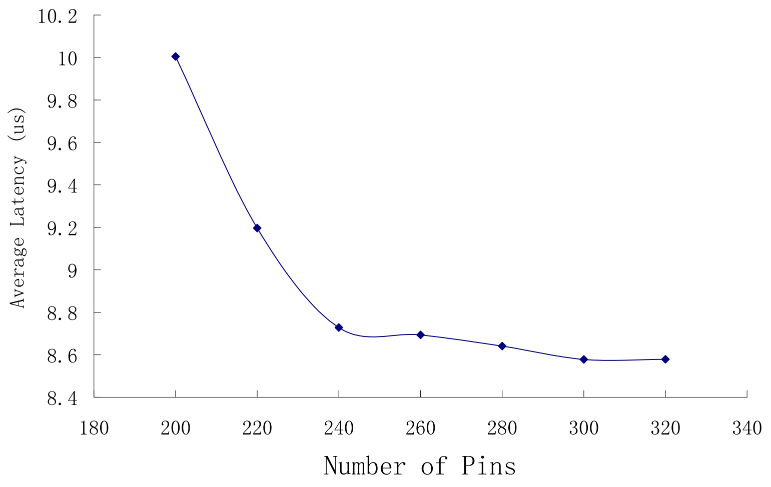


Figure IV.9: Average Latency vs. Number of Pins in a Node Card

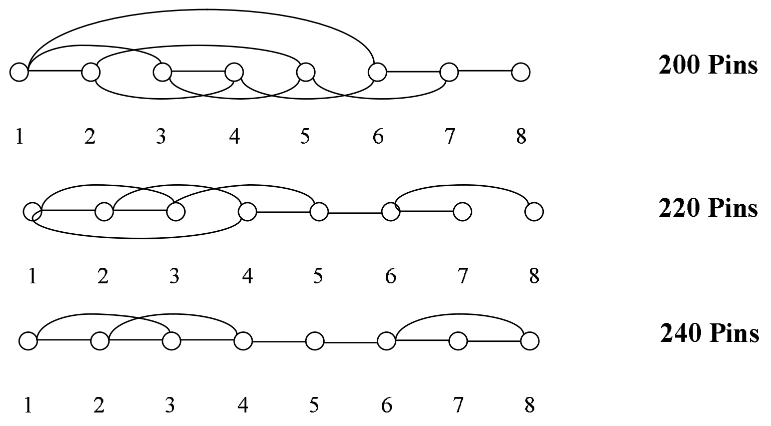


Figure IV.10: Optimal Topologies with Different Number of Pins

Benchmark code	**Size**	**# Proc**	**# Comm**
Block tridiagonal solver (BT)	102^3	121	726
Conjugate gradient (CG)	75000	128	640
LU solver (LU)	102^3	128	466
Multigrid (MG)	256^3	128	920
Pentadiagonal solver (SP)	102^3	121	720

Table IV.3: Characteristics for Five NAS Parallel Benchmarks

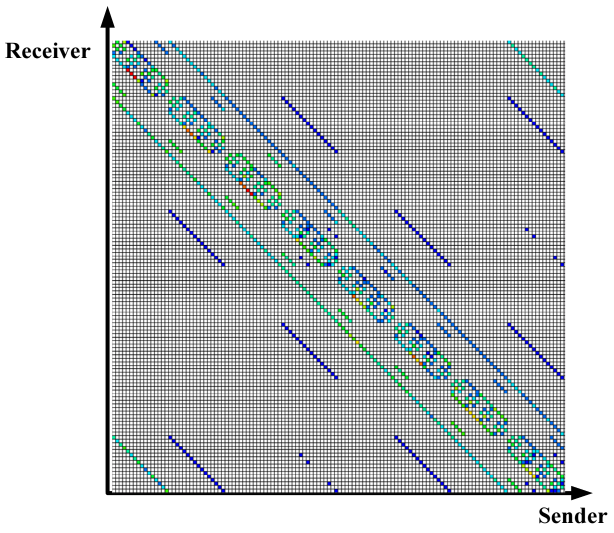


Figure IV.11: The Traffic Pattern for Benchmark Instance MG

We can see that each process communicates with 4 – 6 other processes in average. As an example, Figure IV.11 shows the traffic pattern for the benchmark instance MG, where the color denotes the magnitude of average transfer rate.

Before running the experiments, we use a simulated annealing (SA) algorithm to perform the task placement. The SA algorithm optimizes the estimated latency (i.e. estimating the latency by the distance but ignoring the traffic congestion). Then we feed the communication patterns based on the placement result to our design flow. The results are shown in Table IV.4. The latency values are in the unit of us, and the values in the brackets are the latencies normalized to the "Optimal" column. We run our design flow on the five benchmarks in the following three ways.

Table IV.4: Latency Comparison on NAS Parallel Benchmarks

Benchmark	**Optimal (us)**	**Aggregate (us)**	**Uniform (us)**	**3D Torus (us)**
BT	1.30 (1)	1.44(1.11)	1.67(1.29)	2.00(1.54)
CG	0.96 (1)	1.01(1.05)	1.36(1.41)	1.76(1.84)
LU	1.05 (1)	1.38(1.32)	1.60(1.59)	1.69(1.60)
MG	0.89 (1)	0.90(1.02)	1.15(1.29)	1.65(1.83)
SP	2.24 (1)	2.49(1.11)	2.73(1.22)	3.60(1.61)

First, we assume the interconnection network can be customized for each benchmark, therefore obtain the optimal topology for each benchmark. The latency is shown in the "Optimal" column in Table IV.4. Secondly, only one fixed interconnection network is allowed for all the benchmarks. In this case, we aggregate all five traffic patterns by adding them together and feed into the design flow. After the optimal solution for this aggregate traffic pattern is obtained, it will be evaluated for the five instances separately. The latency values are shown in the "Aggregate" column in Table IV.4. Thirdly, we assume the dimension and pin resources are uniformly distributed to all the wires, which is much less flexible design choices than our design flow. The latency values obtained are shown in the "Uniform" column in Table IV.4. We also obtain the latency results of 3D torus structure, which are shown in the last column.

Table IV.4 shows that the optimal topology found by the design flow can achieve much smaller average latency than the 3-D torus. It also indicates that the aggregate traffic pattern is representative, as the optimal topology identified using the aggregate traffic achieves similar latency values to the individual ones. In addition if we compare the latency values in the "Uniform" column with those in the "Optimal" and "Aggregate" columns, we find that it is essential to enable the non-unform resource allocation for wires, since the flexible choices could accommodate particular traffic patterns and reduce the average latency.

Lastly, to demonstrate the tradeoff between the throughput and latency, we manually increase the communication traffic by multiplying the coefficients

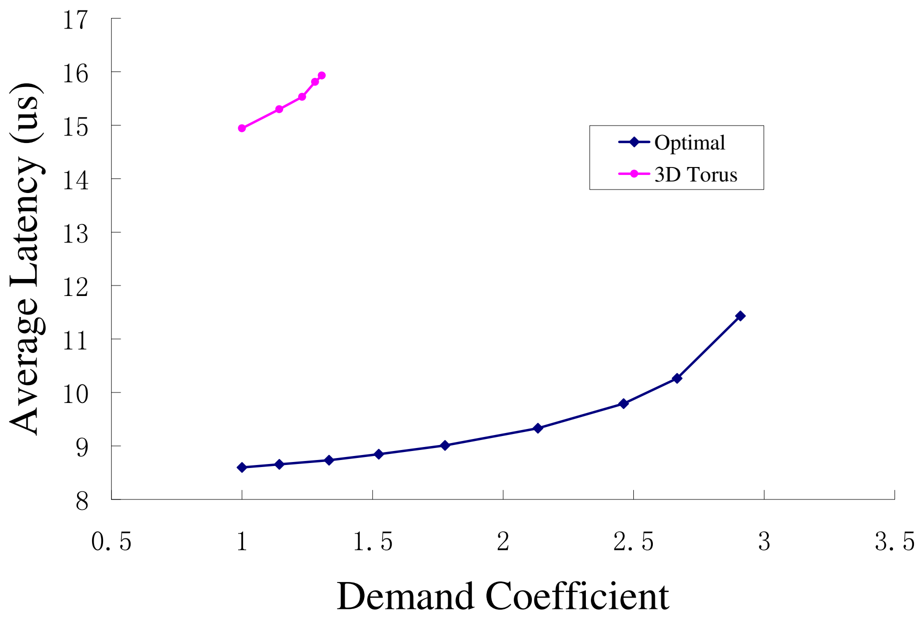


Figure IV.12: Latency-Throughput Tradeoff Curves for the Aggregated Benchmark Instance

to all the communications uniformly. We compute the average latency for the aggregated case. The results are shown in Figure IV.12, the two curves represent the latency values for the optimal topologies selected by the design flow, and the 3D torus topology respectively. Similar to the curves in Figure IV.5, the curve for optimal topologies is also super linear, which indicates that different optimal topologies are selected with the increase of communication demands. Comparing the two curves, we observe that much lower latencies could be achieved when optimal topologies are used. Also, with the flexibility of the topologies, the system is able to accommodate much larger traffic throughput. This again shows the strength of our design flow over fixed topologies.

IV.5 Summary

We propose a design methodology to synthesize the supercomputer interconnection topologies according to the communication patterns, traffic amount, and physical constraints. Users generate a large pool of candidate topologies and provide the physical constraints, and our MCF solver is able to identify the best topology in terms of average latency by evaluating all the topologies. In the example that we demonstrated, we are able to find different good network topologies on a midplane of the Blue Gene/L supercomputer, each of which has better performance than its original 3D torus. In addition, we are able to correctly identify the bottleneck of the communication, and therefore can provide useful information for designers to further enhance the performance.

This chapter includes the contents of one paper "Advancing Supercomputer Performance Through Interconnection Topology Synthesis", by Y. Zhu, M. Taylor, S. B. Baden, C.K. Cheng, to appear in Proceedings of International Conference on Computer Aided Design, 2008. The dissertation author was the primary researcher and co-author of the paper.

V

Conclusion

Motivated by the increasing importance of interconnection networks in both on-chip architectures and computer clusters due to the advancing technology trends, we study the synthesis and optimization of interconnection networks by algorithmic approaches in this dissertation. Two comprehensive case studies, which are Network-on-Chip and supercomputer, are presented to support our study. In the following, Section V.1 will summarize the contributions of this work, and Section V.2 will present several future research directions to extend this study.

V.1 Summary

In this dissertation, we propose a general design methodology to synthesize and optimize interconnection networks among processors, both on-chip and across boards. We explore a large variety of topologies, and design several efficient variations of multi-commodity flow algorithms to solve our problems. Our contributions are summarized as follows:

1. First, we improve the multi-commodity flow algorithms, which are the core component of our methodology. Chapter II presents the detailed contributions from a theoretical aspect. We devise efficient polynomial time approximation schemes to deal with the network synthesis problems raised in

interconnections among processors. Although there were several approximation algorithms available in the literature, they were not able to handle the practical constraints in practice. Our new algorithms can solve the formulations which incorporate these constraints, with proved convergence speed. In addition, we also devise an interval estimation heuristic to speed up the searching process, without compromising the accuracy of algorithms.

2. The Network-on-Chip problem is first investigated from a practical aspect in Chapter III. We study the tradeoffs between NoC power efficiency and average latency. By adopting a MCF formulation, we are able to reduce power consumption of NoC under given latency constraints through simultaneous optimization of network topologies and wire styles. We conduct the experiments for 8×8 NoCs. The results show that power and latency co-optimization is critical in NoC design. With 2% latency overhead, up to 19.4% power savings can be seen. Regarding the comparison among mesh, torus and hypercube topologies, our optimized design can improve power latency product by up to 52.1%, 29.4% and 35.6%, respectively.

3. We also perform a case study on supercomputer interconnection networks in Chapter IV, which is in a different domain but can be solved using our methodology too. We synthesize the supercomputer interconnection topologies according to the communication patterns, traffic amount, and physical constraints. Users generate a large pool of candidate topologies and provide the physical constraints, and our MCF solver is able to identify the best topology in terms of average latency by evaluating all the topologies. In the example that we demonstrated, we are able to find different efficient network topologies on a midplane of the Blue Gene/L supercomputer, each of which has better performance than its original 3D torus. In addition, we are able to correctly identify the bottleneck of the communication, and therefore can provide useful information for designers to further enhance the performance.

V.2 Future Directions

The work done in this dissertation, though having achieved satisfying results, could still be improved in several aspects. We shall mention several possible directions as follows:

1. As mentioned in Section II.4, there is still a large room to improve the performance of the MCF algorithms we discovered. If we are able to find better length update functions or prove a tight bound, the convergence speed would be greatly improved. It will also benefit the practitioners. For example, with the increase in the number of cores in on-chip networks and supercomputers, we need to use more efficient MCF algorithms to facilitate the synthesis and optimization of their interconnection networks. Hence, inventing and improving the algorithms are the common interests of both theorists and practitioners.

2. Regarding the research in NoC, one of the future research directions is to extend this work to the NoC designs where tiles may have different sizes and irregular locations. These NoC architectures enable more efficient design space exploration, but brings much more computational complexity. We will study efficient algorithms for such MCF formulations.

3. Several research directions could be carried in supercomputer research. A direct extension of this work is to take the power consumption into consideration. Designers are interested in topologies that achieve low power consumption along with good performance in terms of latency. Our MCF formulation is easy to modify for this case, with provided accurate wire and router power models. We are going to work on this task and believe that the power-latency tradeoffs are very useful for supercomputer designers. The other direction is to apply this methodology on other supercomputers, e.g., Cray Black Widow, to further show the strengths of our flow. This requires us to find out more physical data about these supercomputers. With more

concrete instances, we may further improve our formulation and algorithms so that it could become a powerful synthesis tool for supercomputer architects.

Bibliography

[1] http://cs.anu.edu.au/ bdm/nauty.

[2] http://public.itrs.net.

[3] http://www.cray.com/products/x1/.

[4] http://www.cray.com/products/xt3/.

[5] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[6] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks. *Technical Report NAS-95-020, NASA Ames Research Center*, 1995.

[7] The BlueGene/L Team. An overview of the BlueGene/L supercomputer. In *The 15th Annual SC conference*, 2002.

[8] R. Chang, N. Talwalkar, C.P. Yue, and S.S. Wong. Near speed-of-light signaling over on-chip electrical interconnects. *IEEE J. of Solid-State Circuits*, 38(5):834–838, 2003.

[9] H. Chen, R. Shi, C.K. Cheng, and D. Harris. Surfliner: A distortionless electrical signaling scheme for speed of light on-chip communications. In *Proceedings of IEEE Intl. Conf. on Computer Design*, pages 497–502, 2005.

[10] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(5):406–424, 1953.

[11] W. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[12] W. Dally. Interconnect-centric computing. *Keynote Speech, IEEE 13th International Symposium on High Performance Computer Architecture*, 2007.

[13] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of ACM/IEEE Design Automation Conf.*, pages 684–689, 2001.

[14] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks.* Elsevier, 2005.

[15] N. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49:758–764, 1946.

[16] A. Gara et al. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2/3):195–212, 2005.

[17] G. Bhanot et al. Optimizing task layout on the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):489–500, 2005.

[18] N. R. Adiga et al. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2/3):265–276, 2005.

[19] P. Coteus et al. Packaging the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):213–248, 2005.

[20] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal of Disrete Math*, 13(4):505–520, 2000.

[21] R. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

[22] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[23] A. V. Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. *Technical Report STAN-CS-91-1372, Department of Computer Science, Stanford University*, 1991.

[24] M. Grigoriadis and L. Khachiyan. Approximate minimum-cost multicommodity flows. *Mathematical Programming*, 75:477–482, 1996.

[25] R. Ho, K. W. Mai, and M. Horowitz. The future of wires. *Proceedings of IEEE*, 89(4):490–504, 2001.

[26] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 233–239, 2003.

[27] T. C. Hu. Multi-commodity network flows. *Operations Research*, 11:344–360, 1963.

[28] Y. Hu, H. Chen, Y. Zhu, A. Chien, and C.K. Cheng. Physical synthesis of energy-efficient networks-on-chip through topology exploration and wire style optimization. In *Proceedings of International Conference on Computer Design, to appear*, 2005.

[29] G. Karakostas. Faster approximations schemes for fractional multicommodity flow problems. In *Proceedings of the 13th Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 166–173, 2002.

[30] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 18–25, 1995.

[31] F. Karim, A. Nguyen, S. Dey, and R. Rao. On-chip communication architecture for oc-768 network processors. In *Proceedings of ACM/IEEE Design Automation Conf.*, pages 678–683, 2001.

[32] S. W. Keckler, D. Burger, C.R. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M.S. Hrishikesh, N. Ranganathan, and P. Shivakumar. A wire-delay scalable microprocessor architecture for high performance systems. In *Proceedings of Intl. Solid-State Circuits Conf.*, pages 1068–1069, 2003.

[33] J. Kim, J. Balfour, and W. Dally. Flatterned butterfly topology for on-chip networks. In *Proceedings of 40th IEEE/ACM International Symposium on Microarchitecture*, pages 172–182, 2007.

[34] J. Kim, W. Dally, B. Towles, and A. Gupta. Microarchitecture of a high-radix router. In *Proceedings of the 32nd International Symposium on Computer Architecture*, pages 420–431, 2005.

[35] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23:466–487, 1994.

[36] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, M. Forsell J. P. Soininen, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *Proceedings of IEEE Symp. on VLSI*, pages 117–124, 2002.

[37] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241–251, 1997.

[38] B. D. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26:306–324, 1998.

[39] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 network architecture. In *Hot Chips 9*, pages 113–117, 2001.

[40] S. Murali and G. De Micheli. Bandwidth constrained mapping of cores onto noc architectures. In *Proceedings of Asia and South Pacific Design Automation Conf.*, volume 2, pages 896–901, 2004.

[41] S. Murali and G. De Micheli. Sunmap: A tool for automatic topology selection and generation for nocs. In *Proceedings of ACM/IEEE Design Automation Conf.*, pages 914–919, 2004.

[42] U. Y. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization vis long-range link insertion. In *Proceedings of Intl. Conf. on Computer Aided Design*, pages 246–253, 2005.

[43] L. S. Peh. Flow control and micro-architectural mechanisms for extending the performance of interconnection networks. *Ph.D. Thesis, Stanford University*, 2001.

[44] L. S. Peh and W. Dally. A delay model for router microarchitectures. *IEEE Micro*, 21(1):26–34, 2001.

[45] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[46] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proceedings of the 6th Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 486–492, 1995.

[47] S. Scott, D. Abts, J. Kim, and W. Dally. The blackwidow high-radix clos network. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 16–28, 2006.

[48] S. Scott and G. Thorson. The cray T3E network: Adaptive routing in a high performance 3D torus. In *Hot Interconnects 4*, 1996.

[49] C. Seitz. Let's route packets instead of wires. In *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pages 133–138, 1990.

[50] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. *Journal of ACM*, 37:318–334, 1990.

[51] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, and F. Ghodrat *et. al.* The raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 2002.

[52] P. M. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[53] H. Wang, L. S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection network. In *Int. Symp. on Microarchitecture*, pages 294–305, 2002.

[54] H. Wang, L. S. Peh, and S. Malik. A technology-aware and energy-oriented topology exploration for on-chip networks. In *Proceedings of Design, Automation and Test in Europe*, volume 2, pages 1238–1243, 2005.

[55] T. T. Ye, L. Benini, and G. De Micheli. Analysis of power consumption on switch fabrics in network routers. In *Proceedings of ACM/IEEE Design Automation Conf.*, pages 524–529, 2002.

[56] N. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.