The Dissertation Committee for Xiang Wu
certifies that this is the approved version of the following dissertation:

Scheduling On-Chip Networks

Committee:

Adnan Aziz, Supervisor

Joydeep Ghosh

Mohamed G. Gouda

David Z. Pan

Peter-Michael Seidel

Scheduling On-Chip Networks

by

Xiang Wu, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2009

To those whose words and deeds invite my thoughts.

Acknowledgments

It all started with a few mathematical puzzles that my adviser—Dr. Aziz posed to me. I showed him my answers on the white board and he challenged more. We kept bouncing ideas off each other for a few weeks. Till one day, he told me that I should solve this network scheduling problem and build it into a Ph.D. dissertation. At that time, little did I know how much work was ahead of me, nor did I feel confident. My adviser simply required me to formalize our discussion and keep everything in a ready-to-publish mode. More time was put into this problem and we got our first publication at Hot Interconnects with very warm reviews. For a moment, I thought I saw light from the end of a long dark tunnel.

So after three years, today I am about to conclude my pursuit of Doctor's degree, I realize how lucky I was to be surrounded by a group of people who never stopped encouraging and supporting me. My adviser always told me how impressive he was with my ability. And when things seemed to halt, he never pressured me without patience. It was his great idea to draw collaborators to our quest that helped me launch my research. His advice on reading and thinking changed my weekend routine and probably much more beyond my spare time. It brought me down to the fundamentals of a researching spirit, absorbing ideas and creating my own. He also offered great counseling

on my career and it would take a much bigger volume for me to enumerate his influence in shaping my view of world and life in general.

I am always grateful for the time that my committee members: Dr. Ghosh, Dr. Gouda, Dr. Pan and Dr. Seidel, put into this work. Every discussion I held with them furthered my understanding of the topic and shed light on new perspectives. Dr. Ghosh and Dr. Pan drove the course project completed by Duo Ding (a Ph.D. student of Dr. Pan) and me to a Best Student Paper Award at International Conference on IC Design and Technology (ICICDT) 2009. All these experiences are certainly instrumental and enjoyable, and our collaboration will last well after the end of my Ph.D. study.

I also need to thank Dr. Higle, who supervised my M.S. degree at University of Arizona. She is now serving as the chairwoman of the Integrated Systems Engineering (ISE) department at Ohio State University. When working on her dedication—higher education and community activity, she is always a vocal and steadfast leader. In person, she is an inspiring mentor, who always keeps an open door for anyone who is seeking guidance.

Over years, we have coauthored several papers with researchers all around the country. Two of Dr. Aziz's former students: Marghoob Mohiyuddin and Amit Prakash kindly shared their great ideas on on-chip communication and DSP algorithms. And Dr. Wolf from Princeton University brought his unique insight into our Asilomar paper. Dr. Massoud's research group at Rice University helped us apply our scheduling algorithms to study reliability issues in deep sub-micron designs. The results out of this collaboration far

exceeded our expectation. His students: Mosin Mondal and Tamer Ragheb conducted extensive study on statistical models of on-chip interconnect delays, which provided the quantitative pillar of our publications. Many of my friends at UT communication group discussed with me about communication and information theory in general. They never seemed to be bored by my babbling in our happy hours. Just to name a few of these great fellows: Wei Wu, Runhua Chen and Taiwen Tang. I also need to mention Dr. Pan's students Duo Ding and Xiaokang Shi, who helped me get started in design for manufacturability (DFM) techniques.

Colleagues at Advanced Micro Devices (AMD) could not be more helping. They knew that I was marching towards Ph.D. since the initial interview. And all of them fondly checked on my progress from time to time. I knew they were showing their support and hoping me the best. The relaxed working environment allows me to take care of business in office and on campus simultaneously. I don't think it would be possible for me to finish this journey if I worked at any other place. I just feel proud to be part of this enterprise.

The lovely city of Austin offers me almost everything I need when I do not want to bury myself with math and programs. The little café inside Borders was my perfect house of Zen. The bookstore was originally located just across my office and later moved into the even more enjoyable Domain mall. I can't remember how much time I have killed by sipping iced tea and gazing into sunset from the its gorgeous grand window. Though I get my food for thought almost exclusively from Borders, the Tapioca house on the

Guadalupe street offers the best food for my body. The tea drinks are fantastic and combos are salty and spicy, which is very similar to my hometown style. And most important of all, it is a place for people to meet and talk. It can be crowded and steamy at times, but it is never short of fun.

Longhorn sports now have become a big part of my life. There are numerous moments of despair and exuberance throughout these years. And "Texas Fight" will be resounding in my heart for the rest of my life. Being a fan, I can feel the connection in the Longhorn family and the pride of championships.

I want to reserve the last part of this acknowledgement to my family. My parents tried their best offering education and nurturing and I could not recall a single occasion, in which they would sacrifice their children's good. My sister has been literally my adviser-of-life ever since we moved to Beijing to attend colleges and I still find her wisdom amazing, especially after my wonderful nephew—Logan was born. I know I can count on them when things are all bad and I feel beaten down. My pursuit would not bear the same meaning if they were not holding my hands against all odds. In this sense, this dissertation is a small gift that I want to present to my family, not for what they have done because it simply outweighs this work, but for what I owe to them as this is the best of me to date.

Scheduling On-Chip Networks

Publication No. _____

Xiang Wu, Ph.D.
The University of Texas at Austin, 2009

Supervisor: Adnan Aziz

Networks-on-Chip (NoC) have been proposed to meet many challenges of modern Systems-on-Chip (SoC) design and manufacturing. At the architectural level, a clean separation of computation and communication helps integration and verification. Networking abstraction of the communication infrastructure also promotes reuse and fast development. But the benefit is most visible when it comes to circuit and physical design. Networks can be made sparse and regular and thus facilitate placement and route. It is also much easier to reach timing and power closure as NoC shield communication details away from complicating analysis. Last but not the least, networks are flexible at the design stage and adaptable post-silicon. Many techniques of tackling process variation and interconnect failure can be built upon NoC.

However, when interconnects are time multiplexed in a NoC, the network's performance will deteriorate if it is not scheduled properly. For a wide range of applications, the traffic on the network can be determined before runtime and offline scheduling offers guaranteed performance and enables simple

design. We propose a synthesis flow that takes the data flow graph of the application and a network topology as inputs; and outputs an offline schedule that can be deployed directly to the NoC. We analyze the complexity of combinatorial problems that arise from this context and provide efficient heuristics when polynomial time algorithms are not available assuming P $\neq$ NP. Results on LDPC decoding and FFT designs are compared with previous ones.

We further apply our findings to parallel shared memories (PSM) and formalize the PSM architecture and its scheduling problem. An efficient heuristic is derived from our algorithm for unbuffered networks. Another application exemplifies how the NoC can be reprogrammed after silicon is back from fab in order to avoid failed interconnects due to process variation. A simple statistical model is studied and the simulation result is rather interesting. We find out that high performance and yield are not always at conflict if we are able to change the network schedule based on silicon diagnosis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In this work, we will investigate a synthesis framework for on-chip networks from both theoretical and practical perspectives. As transistors scale down and switch faster, metal wires start to contribute a significant portion of the delay, power consumption and reliability downgrade of the overall silicon. This change also requires designers to explicitly consider the impact of long wires even at the standard cell or macro level. Techniques such as buffer insertion have been adopted to meet design specification in physical design flows, but now more and more adjustment entails microarchitectural reshuffle, which is time consuming and error prone. Though other ideas such as low swing circuitry also have found their way in particular types of designs, they are not well automated and very hard to verify under current process variation. In summary, deteriorating performance and increasing design challenge have motivated people to seek new methods to engineer interconnect.

Meantime, modern Systems-on-Chip (SoC) have grown explosively in functionality and complexity. Multi-core or many-core systems become popular. Heterogeneous components are integrated together to support novel ap-

plications and cut the cost further down. Intense competition has driven the time-to-market down to an unheard of term. To survive, a design team now relies on its ability to reuse available components and customize their interaction to deliver the product. All these ask for a highly scalable and efficient on-chip communication infrastructure with great design flexibility and ease to verify. Networks-on-Chip (*NoC*) [1] have emerged as a viable solution to overcome the physical limitation of metal wires [2–4] and bridge the design gap from discrete intellectual property (*IP*) blocks to a fully integrated SoC.

A network replaces point-to-point connections (dedicated wires) with a *switch fabric* (wires connected to programmable crosspoints) [5]. A key advantage is that wiring resources can be shared via time-multiplexing, and so the same communication can be implemented with less interconnect. The network interconnect can also be made more regular, thus accelerating the physical design flow. With clearly defined interfaces and protocols, an on-chip network greatly simplifies the integration process of various on-chip components and reduces the verification and testing cycles.

We intend to build networks for our target applications including digital signal processing (*DSP*), multimedia encoding and decoding and scientific computing. The shared characteristics of our applications are:

1. They all feature complicated algorithms that demand an ensemble of processing elements (*PE*) operating in parallel to achieve high throughput;

2. The traffic between PEs can be determined *statically*, i.e., it is known at compile-time rather than at run-time [6].

Consequently, an optimized mapping from algorithms to networks can be computed offline, exploiting a global view of the solution space, of which dynamic (online) schedulers can not take advantage. We present a synthesis framework that takes an algorithm and a selected topology as inputs; it optimally implements the algorithm's communication for the topology. We address two key optimization problems that arise in this context—*placement*, i.e., spatial binding of computations to PEs on the network, *scheduling*, i.e., constructing a detailed cycle-by-cycle scheme for implementing the communication between PEs on the network.

1.1.1 Parallel Shared Memories

With the advance of microprocessor architecture and semiconductor technology, the performance of a computing system is increasingly limited by its capacity of exchanging data between processors and memories. On one hand, memories have been pulled much closer to processors, resulting in large sized caches and integrated memory controllers on die. On the other, multiple memories must be assembled to sustain high throughput, as N independent memories are able to deliver N times the bandwidth of a single one. In many situations, the parallelism is abundant, but it is simply challenging to keep data flowing as fluidly as computation. As a consequence, on-chip networks is naturally employed to connect processors to memories. In such a setup,

each iteration of computation is decomposed into three steps: 1) "gather"—collecting operands from memories, 2) "compute"—processing fetched data and 3) "scatter"—writing results back into memories. This paradigm is proposed for the Merrimac multiprocessor [7] to achieve high throughput. It is also supported in the Intel's latest Larrabee many-core processor [8] for visual computing.

We refer to the combination of processors, memories, and an interconnection network as the parallel shared memory (PSM) architecture. A physical memory only supports a limited number of simultaneous reads and writes. When many processors are accessing the same memory, some of them will have to wait till the memory has finished servicing others. The arbitration of processors' accesses to memories is directly translated into the schedule for the on-chip network in a PSM architecture. We will build a formal model of memory accesses in the PSM architecture. And not surprisingly, the scheduling problem for the PSM architecture is also NP-hard. An efficient heuristic based on our experience with unbuffered NoC is thus proposed for this specific problem.

1.2 Organization

We will begin with formulations of algorithms, networks and schedules etc. in Chapter 2. We show in Chapter 3 an effective heuristic to tackle the placement problem, a pseudo-polynomial time algorithm for a special case—tree topology and a congestion based heuristic for general networks. Results

of our heuristics on practical applications are presented in Chapter 4. Parallel Shared Memory (PSM) architecture is extensively discussed in Chapter 5. An application of overcoming interconnect failure with NoC is presented in Chapter 6. We summarize related work in Chapter 7.

Chapter 2

Formulation

2.1 Algorithms

Algorithms are formalized using the concept of *synchronous data flow graphs* (DFGs) [9, Chapter 2]. In a DFG, vertices correspond to computations such as addition or multiplication, and directed edges denote data dependencies and delays, as illustrated in Figure 2.1.

An algorithm may have tens of thousands of vertices in its DFG representation. Such algorithms are implemented using *folding* [9, Chapter 6], wherein a much smaller number of hardware units are time-multiplexed to implement the desired computation.

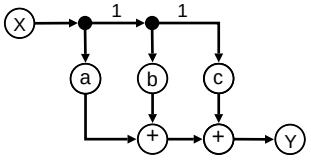

Figure 2.1: An example of DFG: $Y(n) = a \cdot X(n) + b \cdot X(n-1) + c \cdot X(n-2)$. Vertex X is the input and Y is the output. Labeled circles represent multiplications and additions; there are two edges with one sample time delay.

2.2 Networks on Chip

An on-chip network is built using a switch fabric, i.e., a collection of links and programmable crosspoints. The network connects a set of *source nodes* S to a set of *sink nodes* T [5]. We represent the switch fabric as an undirected graph $G = (V, E)$. Sources, sinks and intermediate crosspoints all correspond to vertices, and links are modeled as edges. Intuitively, sources and sinks correspond to processing elements, while edges and intermediate nodes constitute the network connecting PEs. We will refer to a switch fabric and its graph interchangeably.

Our approach can be applied to arbitrary network topologies. Our experiments focus on networks organized as meshes, i.e., with the exception of vertices on the boundary, each PE has links to 4 other adjacent PEs. We consider such networks because they offer an appropriate compromise between connectivity and cost [10].

2.3 Traffic Matrices

A *traffic matrix* is an $|S| \times |T|$ matrix M, where M_{ij} is a non-negative integer encoding the number of packets to be transferred from source i to sink j. Given a fabric and matrix M, a *schedule* is a collection of *configurations*, where each configuration consists of choices for all programmable crosspoints. These choices result in a set of *channels* that connect a subset of S to a subset of T. We assume the fabric does *not* buffer packets internally; hence for a configuration to be *valid*, no two channels can intersect each other. For

each configuration, a fixed-duration *cycle* is allocated to program the fabric and transfer packets. A schedule Σ is said to *complete* the matrix M, if by following the procedure above for each configuration in Σ, we can transfer all packets encoded in M from S to T.

A *workload* for a switch fabric is defined to be an ordered set of traffic matrices that the fabric needs to implement during the computation. A schedule Σ is said to complete a workload if by carrying out Σ cycle by cycle, all matrices in the workload will be completed in order.

We employ a straightforward transformation from a DFG to a workload. Note that each edge in the DFG will result in a packet to be delivered by the network if its both ends are not mapped to the same PE. When an edge's packet has not been transferred, we mark this edge as *unfinished*; and after transferring the packet, we mark it as *finished*. We keep track of the set R of *ready* edges that are marked as unfinished and have no precedent unfinished edges. At any time, set R includes all packets available for scheduling. Therefore, at the beginning of each cycle, we formulate a traffic matrix encoding exactly those packets in R and submit it to our algorithm. At the end of each cycle, since some edges' packets are delivered, some new edges may become ready and should be added into R, and finished ones will be removed from it. A new matrix will then be formulated based on the updated R in the next cycle and we repeat these steps till we finish communication. For ease of exposition, we will focus on traffic matrices from now on.

2.4 Scheduling a Traffic Matrix

Recall that a valid configuration is a set of non-intersecting channels, which corresponds to a collection of paths in G, and no two paths share a common vertex; we refer to such paths as being *vertex disjoint*.

Given a switch fabric G and an assignment of DFG vertices to vertices in G, a matrix m is defined to be *G-feasible* if there exists a single configuration that completes m. It follows that a matrix m is feasible iff all entries in m are either 0 or 1, and all source-sink pairs corresponding to 1s in m can be connected by a collection of vertex disjoint paths in G. Note that each configuration in the schedule can be mapped to a feasible matrix, or equivalently a *vertex-disjoint-path-sets* (VDPS). Consequently, we will interchangeably refer to a schedule as a collection of feasible matrices or a collection of VDPS.

2.4.1 A Nontrivial Example

We present a small but surprisingly interesting instance of the general scheduling problem in Figures 2.2, 2.3 and 2.4. Specifically, it illustrates that building the schedule greedily—that is by always picking the largest possible VDPS—is suboptimum.

2.5 Physical Implementation

Our development is at a relatively high level of abstraction, compared to the final gate-level implementation of the network. For example, the cycle we refer in preceding sections to is not the chip's clock period T_{clk}; it is the

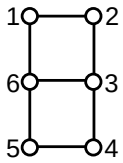

Figure 2.2: A mesh-structured switch fabric G. Each vertex can be either a source or a sink, but not both, in a cycle.

$$\begin{pmatrix} 0 & 1^a & 0 & 1^b & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1^c & 0 & 1^d \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1^e & 0 & 0 & 0 & 1^f \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2.3: Traffic matrix M for the fabric in Figure 2.2. The superscripts are packet identifiers, e.g., we will refer to the packet from Source 1 to Sink 2 as a.

time to transfer the packet. This time is significantly larger than T_{clk}, and for this reason we model all channels as having the same delay, even though in practice longer interconnects may be pipelined, thereby inducing a latency of a few chip clock periods.

Our primary focus in this chapter is to compute an optimized placement and schedule for a given network topology and application. The broader design problem needs to consider the VLSI implementation cost of the network. The implementation cost of a network can be estimated using predictive modeling theory for VLSI. The network is physically realized using buffered wires, and programmable crosspoints. The area, delay, and power of an optimized inter-

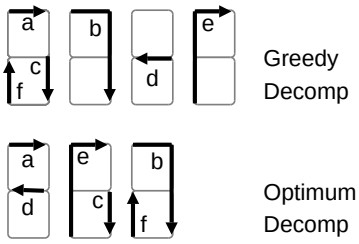

Figure 2.4: Greedily constructed and optimum schedules for G and M as presented in Figure 2.2 and 2.3, respectively. The largest VDPS corresponds to the packets $\{a, c, f\}$, but selecting it leads to a schedule that takes 4 cycles.

connect of a given length in a given manufacturing process can be estimated using existing techniques, e.g., [11–13]. Similar values for crosspoints can be derived using the estimation approach in [4, Chapter 4].

Chapter 3

Scheduling Unbuffered Networks

3.1 Spatial Binding

3.1.1 Incremental Algorithm

A spatial binding consists of mapping rows and columns of a traffic matrix M to fabric nodes such that the final schedule produced has the minimum number of cycles. We will refer to the same mapping also as "placement". It is shown in [14] that calculating the optimum schedule for any given placement is NP-hard. Consequently, it is very difficult to evaluate how good a placement is. Hence we design a different objective function that is much easier to compute and very helpful to the later scheduling step. Specifically, given a placement Π, define $d_\Pi(s,t)$ to be the shortest distance from s to t when all edges are of unit length. The objective function $Z(\Pi)$ we use is:

$$Z(\Pi) = \sum_{s \in S} \sum_{t \in T} M_{st} \cdot d_\Pi(s,t)$$

For a mesh fabric, $d_\Pi(s,t)$ is the Manhattan distance and can be calculated in constant time. Furthermore, if we incrementally update the placement, we just need to calculate the reduction in Z caused by the incremental update.

The *weight* function $w(x,y)$ between two elements x and y in the set $U = S \cup T$ is defined to be $M_{xy} + M_{yx}$. Starting from an arbitrary placement, we

make improvement by continuously applying *exchange* operations till we can not improve Z anymore. An exchange is an operation on two elements in U, in which we swap the nodes that they are mapped to under current placement. Clearly, after the exchange of x and y under Π, we obtain a new placement Π' satisfying (1.) $\Pi'(x) = \Pi(y)$ and $\Pi'(y) = \Pi(x)$ and (2.) $\Pi'(u) = \Pi(u)$ if u is not x or y. To ensure finite termination, we perform the exchange only when the operation results in a *strictly* smaller objective $Z(\Pi') < Z(\Pi)$. Specifically for x and y from U, we need to calculate the following quantity:

$$Z(\Pi) - Z(\Pi') = \sum_{u \in U} [w(u,x) - w(u,y)][d_\Pi(u,x) - d_\Pi(u,y)]$$

We perform the exchange iff $Z(\Pi) - Z(\Pi') > 0$.

The placement heuristic is presented in detail in Algorithm 1. We point out that in Step 7, only when $w(u,x) > 0$ or $w(u,y) > 0$ do we have an item to accumulate. In most situations the matrix is sparse, i.e., the number of non-zero entries in a row or column is $O(1)$. The time taken in Step 7 is therefore $O(1)$ as well, very fast in practice. For networks up to a few hundred nodes, our algorithm runs for less than 30 minutes. For networks of even larger sizes, we may either terminate the run because of timeout or divide the large network into small blocks and perform the algorithm within each block respectively.

3.1.2 Example

We illustrate the effectiveness of our heuristic by placing the matrix in Figure 3.1 on a 4×4 mesh. Our heuristic transforms a starting placement that is very inefficient to an optimal one as shown in Figure 3.2.

Algorithm 1 Placement Heuristic

Input: graph G, set $U = S \cup T$ and matrix M
Output: Π—a mapping from U to vertices in G

```
1: Initialize Π arbitrarily;
2: Calculate the weight w(x, y) for all elements in U;
3: repeat
4:    Improved ← False;
5:    for all x ∈ U do
6:       for all y ∈ U do
7:          Calculate Δ = Z(Π) − Z(Π′) for x and y;
8:          if Δ > 0 then
9:             Map x to Π(y) and y to Π(x);
10:            Improved ← True;
11:         end if
12:      end for
13:   end for
14: until Improved = False
```

$$\begin{pmatrix} 0 & 1 & 1 & 2 & 2 & 0 & 2 & 0 \\ 1 & 2 & 2 & 1 & 0 & 2 & 2 & 2 \\ 2 & 1 & 1 & 0 & 0 & 2 & 2 & 0 \\ 1 & 0 & 2 & 0 & 1 & 1 & 0 & 2 \\ 0 & 0 & 2 & 2 & 2 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 & 2 & 0 & 0 & 1 \\ 2 & 2 & 1 & 2 & 2 & 1 & 1 & 1 \\ 0 & 2 & 2 & 0 & 2 & 0 & 0 & 2 \end{pmatrix}$$

Figure 3.1: Traffic matrix as the input of Algorithm 1.

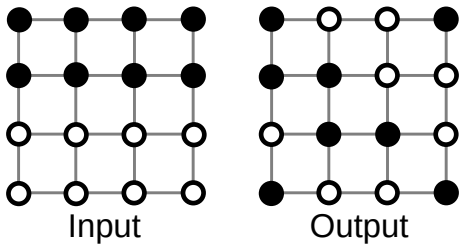

Figure 3.2: Input and output placements of Algorithm 1 based on matrix in Figure 3.1; solid circles represent sources, empty circles represent sinks, they are both placed on a 4×4 mesh.

3.1.3 Extension to Temporal Binding

In the previous spatial binding step, we assign each calculation in the data flow graph to a discrete PE. In reality, it is not always possible or optimal to dedicate a PE to a single calculation throughout the algorithm. More precisely, suppose there are N calculations in the DFG, we may choose to carry them out with N/k PEs. The end result is that it will take longer to produce each result, but in a fully pipelined design, the throughput will be much less affected even with only $1/k$ of computing resources *if* we can efficiently time multiplex PEs and relay intermediate results timely.

Algorithm 1 can be easily extended to calculate temporal binding if there is no dependency among packets. We need to assign k calculations, i.e., k of the rows and columns to a vertex. The overall objective function $Z(\Pi)$ is unchanged. But the atomic exchange operation now only swaps a single calculation from a vertex to another calculation of the other. From a communication centric point of view, this procedure yields improvement even

in the face of interdependency among packets because computations that need to communicate to each other are pulled together.

3.2 Fast Algorithm for Tree Networks

In this section we present a pseudo-polynomial algorithm that solves the scheduling problem exactly for switch fabrics of which the graph G is a tree; we will refer to such fabrics as *tree fabrics*.

Before presenting the algorithm, we define a few useful functions. The *incidence function* $\chi(p, q, v)$ is 1 if the path from source p to sink q includes v and 0 otherwise. The function χ is well defined because p and q uniquely decide the path connecting them in G. We define the *load function* L_v as $L_v = \sum_{p,q} M_{pq}\chi(p, q, v)$. We will also use the *level* of a vertex extensively in the algorithm. Given an arbitrarily chosen root vertex, the level is defined as the distance obtained by a breadth-first search from the root. Given a path P, if v appears in P and *level*(v) is the minimum among all vertices in P, we say P *roots* at v. Evidently each path has one and only one root.

Theorem 3.1. *Given a tree fabric G and traffic matrix M, Algorithm 2 returns an optimum schedule in time Poly(n, A), where n is the size of G and A is the entry in M with the largest value.*

First, we make the claim, proved in Lemma 3.1, that we can always find such a P in Line 9 of Algorithm 2. For now simply note that when a VDPS is generated, the set S with the largest load is removed from T_l. Even though

Algorithm 2 Scheduling on Tree Fabrics

Input: tree G and matrix M

Output: $Sched$ — a collection of vertex-disjoint-path-sets

1: Build a table T_l mapping a load value l to the ordered set $S = \{v | L_v = l\}$; T_l is sorted using key l descending; S is sorted using key $level(v)$ ascending;
2: Build a table T_p mapping a vertex v to the set of paths that root at v (T_p includes all paths needed to discharge the traffic);
3: $Sched \leftarrow \emptyset$;
4: **while** T_l is not empty **do**
5: Pop the first (maximum load) set S from T_l;
6: $VDPS \leftarrow \emptyset$;
7: **while** S is not empty **do**
8: Pop the first (minimum level) vertex v from S;
9: Pop one path P rooting at v from T_p;
10: Add P to $VDPS$;
11: **for all** p in P **do**
12: **if** $p \in S$ **then**
13: Remove p from S;
14: **else**
15: Remove p from $T_l[L_p]$;
16: **end if**
17: $L_p \leftarrow L_p - 1$;
18: **if** $L_p > 0$ **then**
19: Add p back into $T_l[L_p]$;
20: **end if**
21: **end for**
22: **end while**
23: Add $VDPS$ to $Sched$;
24: **end while**

some vertices in S will be inserted back, their loads will be decreased by 1 as in Line 17, which means the largest load in table T_l will be decrease by 1 in each iteration of the outermost while loop. This guarantees a finite termination of the algorithm. To be more precise, the main while loop will iterate exactly $\max\{L_v\}$ times, which is bounded by $O(n^2A)$. All operations inside the loop are clearly polynomial time, hence the time bound in Theorem 3.1 is confirmed. One byproduct is that the size of the schedule *Sched* is proved to be $\max\{L_v\}$, which is the least number of cycles possible because all paths passing that maximum loaded vertex must be used in distinct cycles. Technically, because of the dependence of run-time on the entries of M, the algorithm is pseudo-polynomial. In practice, the entries in M are small, thus the algorithm is truly polynomial time.

To complete the proof of Theorem 3.1, we need to prove the following lemma:

Lemma 3.1. *When Line 9 of Algorithm 2 is encountered during execution, there always exists a path P rooting at v, which is the vertex of the minimum level in S at that moment. Furthermore, such P is vertex disjoint from any other path already added to VDPS.*

Proof. We will induce a contradiction by assuming no such path exists. Based on the assumption, we know that all paths in T_p that include v do not root at v, which means they all include the unique parent of v. Denote this parent vertex as u. This tells us that $L_u \geq L_v$ and $level(u) = level(v) - 1 < level(v)$.

Note that v is in the set of maximum load with the minimum level. And now we have another vertex in the same set with a strictly smaller level. This is a contradiction.

Now that we have existence secured, we need to show that any P rooting at v will not overlap with paths already added into VDPS. If there is a path Q in VDPS and a vertex x appears in both P and Q, another contradiction will be shown right next. Now look at the root of Q, i.e., vertex r. First, $level(r) \leq level(v)$ because otherwise v would be popped out before r. Second, v does not show up in Q because otherwise it would have been removed at Line 13 and cannot be popped out from S. Now clearly $v \neq r$ and we shall build the path from r to x, which is part of Q. The first segment is from r to v and vertices in this part all have levels $\leq level(v)$ obviously. The second segment is from v to x. Since x is below v as P roots at v, all vertices except v in this part will have levels $> level(v)$. Looking at the levels, we can instantly see these two segments do not overlap and combined together they form the unique path from r to x in the tree. This shows that v is actually on the path from r to x. A conflict emerges because that is to say v appears in Q. □

3.3 Heuristics for General Networks

The scheduling problem for a general network is NP-hard [14]. Although there are fast algorithms for crossbars and tree topologies; a crossbar, being dense, has a high implementation cost, and a tree fabric is inadequate because of its limited connectivity [5]. We will see a network organized as a

mesh, which has low implementation cost, performs almost as well as a full crossbar for our applications.

Our heuristic is built upon the metric of *congestion* on edges. Consider an instance of the scheduling problem, with G the fabric, and M the traffic matrix, with sources S and sinks T. Define the distance $d_{e,w}$ between an edge $e = \{u, v\}$ and a vertex w by $\max\{d(u, w), d(v, w)\}$, where $d(x, y)$ is length of the shortest path in G between x and y, when edges are of unit length.

The formula of congestion can be as simple as setting $C_e = 1$, for all edges e. We name this basic version *uniform* congestion, as it does not vary over the edge set. However, the final version we adopt is defined by the following equation:

$$C_e = \sum_{s \in S} \frac{W_s}{d_{e,s}} + \sum_{t \in T} \frac{W_t}{d_{e,t}}$$

where W_s and W_t are the corresponding row and column sums for M. We refer to this one as *distance-inverted* congestion.

The congestion metric is designed to reflect the attenuating trend when the distances to sources or sinks increase. Also importantly, this metric is fast to calculate using breadth-first search from each source and sink: simply accumulate all source or sink quotients without caring about the order in which vertices are visited. Furthermore, the distance part can be cached since when computing the congestion, the topology is always the original graph G.

Several key points in the heuristic are:

1. Loop from Line 9 to 12 chooses the path with the least blockage, thereby avoiding congested regions.

2. The computation of shortest paths in Line 8 with Dijkstra's algorithm is very fast in theory and practice.

3. All vertices in path $p*$ are isolated as in Line 21. This guarantees the vertex disjoint property of all paths added.

4. At Line 25, we always zero out a row or column with the largest sum in M and repetition of this operation will eventually turn M into an all-zero matrix, guaranteeing finite termination.

5. This procedure is constructive and always produces a feasible schedule regardless of the given placement and topology.

The objective here is to minimize the number of transfer cycles, which is not necessarily proportional to the actual packet latency as stated in Section 2.5. When operating in high frequency, the network must be carefully pipelined to match fast PEs, which makes the length of each transfer cycle variable. To account for the pipeline effect, a meaningful extension to Algorithm 3 would be to pack as many short paths as possible into one cycle when we have chosen a long path.

Algorithm 3 Heuristic to Generate a Schedule

Input: graph G, placement Π and matrix M
Output: Σ—a schedule completing M

1: $\Sigma \leftarrow \emptyset$;
2: **while** M has positive entries **do**
3: Backup M in M';
4: $VDPS \leftarrow \emptyset$;
5: Calculate C_e for all edges by doing breadth-first search from each source and sink (rows and columns in M);
6: **repeat**
7: Pick the row or column in M with the largest sum, record the corresponding vertex as $v*$;
8: **if** $v*$ is a source (row chosen) **then**
9: Put all sinks into the target set *Target*;
10: **else**
11: Put all sources into *Target*;
12: **end if**
13: Compute shortest paths $P = \{p_{v*w}\}$ from $v*$ to vertices w in *Target*;
14: **for all** $p_{v*w} \in P$ **do**
15: Determine the blockage of the path, the largest C_e among edges in p_{v*w};
16: Keep track of the path with the least blockage in $p*$ connecting $v*$ to $w*$;
17: **end for**
18: **if** the entry in M for $v*$ and $w* > 0$ **then**
19: Add the path $p*$ to $VDPS$
20: **for all** v in $p*$ **do**
21: Remove all edges incident at v;
22: **end for**
23: Set the row or column for $w*$ to zeros;
24: **end if**
25: Set the row or column for $v*$ to zeros;
26: **until** all entries in M are zeros
27: Add $VDPS$ to Σ;
28: Restore M from M';
29: Determine the number of packets to transfer through each path in $VDPS$;
30: Decrease entries in M according to Step 29;
31: **end while**

Chapter 4

Heuristics Evaluation

As mentioned Section 2.1, there are an enormous number of vertices in the DFGs for the examples that motivated our work—far in excess of the number of physical PEs that can be implemented on a chip. For LDPC decoding, folding (cf. Section 2.1) results in multiple code and check nodes getting mapped to the same PE, which essentially forms a new random traffic matrix of a smaller size. A large size FFT is thus implemented by computing a series of of smaller size FFT, e.g., Maharatna *et al.* [15] implement a 64-point FFT using two 8-point FFT units.

We applied our synthesis flow to implementing LDPC decoding and FFT on a network organized as a mesh. In view of the comments above, we chose to report results on an LDPC block of size 96, and a 512-point FFT.

4.1 LDPC Decoding

An LDPC code is a block code, where there are C bits per block, which include D parity checks. It is most naturally represented as a bipartite graph on a set of C code nodes and D check nodes. The decoding algorithm [16] involves iterations of message passing back and forth between connected code

Code	C1	C3	C3	C4	C5	C6	C7	C8
Num. Cycles	18	16	16	14	14	15	18	17
Lower Bound	13	15	12	12	13	11	12	13

Table 4.1: Size of schedule generated by Algorithm 3 for LDPC codes C1–C8.

and check nodes, and it is this communication that defines the traffic matrix.

We created 8 LDPCs C1–C8 with 96 code and 48 check nodes using randomized code construction techniques [17]. Each entry in the connection matrix corresponds to exact one transfer of a result from a code node to a check node or vice versa. The 144 code and check nodes are embedded on a 23×23 mesh. They are first placed on a 12×12 smaller mesh with our placement heuristic and then we add one extra track between adjacent rows or columns to help routing, which makes the mesh a square of size $23 = 12 \times 2 - 1$. Results for the these 8 different LDPC codes are presented in Table 4.1, Row 2.

4.2 FFT

An N-point FFT can be implemented with parallel hardware using $1 + \log_2 N$ stages, where each stage consists of $N/2$ PEs that implement "butterfly" operations in parallel; the results of these operations are passed on to specific processing elements in the next stage [18]. Specifically, between stage l and $l+1$, $\mathrm{PE}_i(l)$ passes its results to two PEs: (1.) $\mathrm{PE}_i(l+1)$, and (2.) depending on i, either $\mathrm{PE}_{i+2^{l-1}}(l+1)$ or $\mathrm{PE}_{i-2^{l-1}}(l+1)$. It is straightforward to encode the results that need to be communicated from one stage to the next as a

Stage	F1	F2	F3	F4	F5	F6	F7	F8
Num. Cycles	3	4	6	9	3	4	6	9

Table 4.2: Size of schedule generated by Algorithm 3 for each of the 8 stages in a 512-point FFT.

traffic matrix. Since each PE has two inputs and produces two outputs, there are $N/2$ PEs per stage.

We placed 512 PEs on a 63×63 mesh for a 512 point FFT. Half of them implement stage l, and the other half implement stage $l + 1$. The remaining $63^2 - 512 = 3457$ crosspoints on the mesh are used as routing resources. We give the results of our heuristic on the 512 point FFT in Table 4.2.

4.3 Quality of Results

4.3.1 Runtime

For both FFT and LDPC, our heuristic computed the schedule in seconds. Our implementation of the heuristic is very straightforward, and it could likely be sped-up greatly, but there is little incentive to do so since the computation is off-line.

4.3.2 BvN bound

A network is said to be *rearrangeable* if it allows any source to be connected to any sink, regardless of other source-sink connections. For such a network, the minimum number of cycles needed to complete a traffic matrix is the maximum of the row and column sums of the matrix [19]. This value

DFG	F4	F8	C1	C2	L1	L2
DIC	9	9	18	16	32	29
UC	11	12	20	18	37	34

Table 4.3: Examples F4, F8, C1, and C2 are as before; L1 and L2 are larger LDPC codes. The row labeled DIC shows the number of cycles produced by the heuristic based on distance-inverted congestion. The row labeled UC shows the number of cycles produced by the heuristic based on uniform congestion.

is a lower bound on the number of cycles needed to complete the matrix for any network. We calculated this bound for the LDPC traffic matrices. The comparison of our solutions and the optimum bounds is presented in Table 4.1, Row 3. Our schedules are fairly close to the bound, reinforcing our confidence in the heuristic. (Note that a rearrangeable network is quite expensive to implement.)

4.3.3 Distance Inverted Congestion

Table 4.3 demonstrates the extra complexity in distance inverted congestion does result in *consistent* improvement in benchmarks.

4.4 Placement

Rather than measuring the benefits of our placement heuristic against random placements, we illustrate the effectiveness of our placement heuristic by manually generating what we believe to be a reasonably good placement for LDPC. Specifically, we think an *interleaving* placement, in which we place all code nodes at locations $\{(6i, 2j), (6i+4, 2j)\}$, all checker nodes at locations

Code	C1	C3	C3	C4	C5	C6	C7	C8
Heuristic	18	16	16	14	14	15	18	17
Interleaving	30	28	28	18	31	27	29	30

Table 4.4: Schedule sizes for LDPC codes C1–C8 for interleaving placements and those generated by with Algorithm 1.

$\{(6i + 2, 2j)\}$, where $0 \leq i \leq 3$ and $0 \leq j \leq 11$, would be a good placement. In this placement, each checker row comes in between two coder rows and again we reserved one extra track between adjacent rows and columns. This placement seemed to offer reasonable connectivity given the construction of the LDPC matrix.

We ran our placement heuristic against this starting point, and were surprised to find that our placement heuristic generated a placement that when input to our scheduling heuristic resulted in a schedule that reduced the number of cycles by almost 50% compared to the schedule that our scheduling heuristic produced on the original interleaving placement (Table 4.4).

Chapter 5

Application I: Extending to Buffered Networks

5.1 Background

Modern SoCs are increasingly based on multiple processors. These processors have local private memories and share a global memory. The global memory can be used to implement mass communication between processors. To keep bandwidth to the global memory from becoming a limiting factor in performance, the shared memory can be implemented as a collection of on-chip RAMs operating in parallel—the aggregate bandwidth of R parallel RAMs is R times the bandwidth of a single one.

The processors may be connected to the RAMs in a number of ways; the logic implementing the connectivity is known as an *interconnection network*. The most straightforward approach is to use a crossbar—a matrix of pass-transistors [20], and associated control logic. For large numbers of processors and RAMs, the area and delay of a crossbar become prohibitive, and it is natural to consider a sparser interconnection network.

We refer to the combination of processors, memories, and an interconnection network as a parallel shared memory (PSM) architecture. The PSM architecture is the basis for emerging multiprocessor architectures such as the

SUN T2 [21], and has been used successfully in the implementation of high-performance packet routers [22–24],

Each individual RAM can support only a small constant number of simultaneous reads and writes; for simplicity, we assume only one read or write can be performed at a time. Therefore, if two processors simultaneously access a given RAM, one will "stall"—that is wait until the RAM has completed the request for the other processor. It is important to *arbitrate* read and write requests intelligently to minimize stalls. And with a NoC as the communication backbone, memory access arbitration is essentially on-chip network scheduling.

For a large class of programs—e.g., those in DSP, media processing, and scientific computing—the computation is data-independent. This implies that the set of memory requests for all processors is known offline, and hence the scheduling problem can be solved before execution. In this chapter, we focus on performing offline scheduling.

The remainder of this chapter is structured as follows: We formalize the scheduling problem for the NoC in the PSM architecture in Section 5.2. We show that computing an optimum schedule is computationally intractable in Section 5.3. We develop an efficient and accurate heuristic for computing the schedule for the general PSM setup in Section 5.4 and report our experiment results in Section 5.5. We conclude with a summary of contributions and avenues for future work in Section 5.6.

5.2 Scheduling Buffered NoC

5.2.1 Graph-based Representation of NoC

We represent a NoC using an undirected graph $G = (V, E)$. The vertex set V is partitioned into sets P and R, where P corresponds to processors and R to RAMs. An edge exists between $p \in P$ and $r \in R$ exactly when a link exists between processor p and RAM r. (We assume connections are bidirectional, hence G is undirected.) In our model, there are no connections from processors to processors, or from memories to memories. We shall refer to the graph for a given NoC of a PSM architecture as its *PSM graph*. PSM graphs are quite general, and topologies such as meshes and trees can be modeled in this formulation.

We assume that processors communicate with each other using fixed length packets. Given a PSM graph $G = (V, E)$, with P as above, a *traffic matrix* M for G is a $|P| \times |P|$ matrix with nonnegative integer entries. Intuitively, $M_{i,j}$ encodes the number of packets that Processor i has to send to Processor j. We assume henceforth that G is connected, i.e., that any processor can send a packet to any other processor (albeit possibly with many intermediate reads and writes).

The traffic matrix definition is motivated by the computing paradigm used by the Merrimac multiprocessor, which was developed for scientific applications that demand very high performance [7]. Computation in Merrimac is based on iterating through the following three phases: "gather," in which each processor assembles the operands it needs from the global memories,

"compute," in which processors perform arithmetical operations, and "scatter," in which processors write to memories the operands that are required by other processors. The gather and scatter phases implement the communication required between the processors. In the absence of data dependencies, the communication implemented by the gather and scatter phases can be characterized by traffic matrices.

5.2.2 Formalization

5.2.2.1 Scheduling Crossbars

Given a PSM graph G and a traffic matrix M, it is desirable to compute an *optimum schedule*—one that implements the fastest possible transfer of packets specified by M—subject to the constraints of the topology G.

Let the maximum over all row and column sums of M be M_{max}—this is referred to as the Birkhoff-von Neumann (BvN) bound of M. Suppose $|P| = |R|$, and that the connections between processors and memories are implemented by a crossbar. From the results of Chang *et al.* [19], it follows that an optimum schedule can be offline computed, which completes all the packet transfers with exactly M_{max} steps, where each step consists of packets being written to and then read from memories.

5.2.2.2 Basic Definitions

A *cycle* is a fixed amount of time that it takes to transfer a packet via a single link in the network. It is also the smallest unit to measure time

in our model and thus each time instance is uniquely specified by its cycle number. For a packet encoded in the traffic matrix M, we need to assign it a *path*, which is a time sequence of $t+1$ items: $\langle v_s, v_{s+1}, \cdots, v_{s+t} \rangle$. The packet becomes available for dispatch during cycle s at vertex v_s, and it is scheduled to reach its destination during completion cycle $s+t$ at v_{s+t}. During any cycle $s+i$ between s and $s+t$, the packet stops at vertex v_{s+i}. For a path to be *valid*, during cycle $s+i$, we can either buffer the packet at v_{s+i-1}, which implies $v_{s+i} = v_{s+i-1}$; or we can send the packet across a link in the network, which tells us that the sending vertex v_{s+i-1} and receiving vertex v_{s+i} must be connected by an edge in graph G, i.e., $(v_{s+i-1}, v_{s+i}) \in E$. When a vertex is sending or receiving a packet, we call it *active*. Based on the operational nature of processors and RAMs, we assume that each vertex in the graph can be either sending or receiving one packet during any cycle, but not simultaneously. Two paths are *compatible* if during any cycle, they either do not stop at the same vertex or the vertex that they share is not active for both of them. Also note that the number of packets that a vertex can buffer is limited by the size of its local memory or its RAM.

A schedule is a collection of paths, one for each packet encoded in M. It is valid when it satisfies following constraints:

- Every pair of paths in the schedule is compatible.
- No vertex in the graph ever buffers more packets than its memory has capacity for.

Every schedule has a completion cycle, which is the maximum of completion cycles of all packets; an *optimum* schedule is one with the smallest completion cycle.

5.3 Optimum Scheduling is Intractable

We now prove that optimum scheduling of a traffic matrix on the NoC is NP-hard. Specifically, we reduce CNF SAT to the *Disjoint Connecting Paths* (DCP) problem for *k-level* digraphs (defined below), and then reduce the DCP problem to optimum scheduling.

A *k-level* digraph is a digraph $G = (V, E)$ together with a partition $\{V_1, \ldots, V_k\}$ of V such that its edge set $E \subseteq \cup_{i=1}^{k-1}(V_i \times V_{i+1})$. Figure 5.1 shows a simple 3-level digraph.

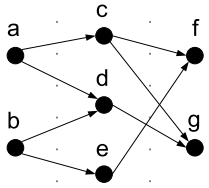

Figure 5.1: A 3-level digraph.

A set of q paths $\{P_i\}_{i=1..q}$ is defined to be *vertex disjoint* if for all $i \neq j$, paths P_i and P_j do not intersect. The *Disjoint Connecting Paths* (DCP) problem consists of a graph G and a set of q disjoint vertex pairs $\{(u_i, v_i)\}_{i=1..q}$; the question is whether there is a vertex disjoint path set $\{P_i\}_{i=1..q}$ such that P_i connects u_i to v_i for all i. The DCP problem for a general graph is known

to be NP-complete [14].

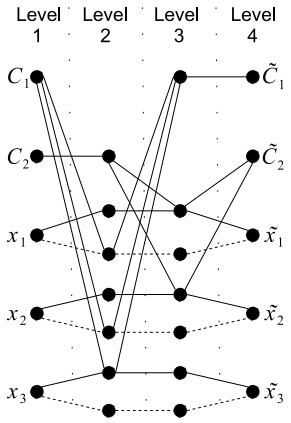

Figure 5.2: A 4-level digraph represents the clause database $C_1 \cap C_2$, where $C_1 = x_1 + x_2 + \overline{x_3}$ and $C_2 = \overline{x_1} + \overline{x_2}$. From left to right, level 2 is reserved for C_1 and level 3 for C_2. Directions on all edges are also from left to right.

Theorem 5.1. *The DCP problem for a k-level digraph is NP-hard.*

Proof. The general method to construct the k-level digraph from a CNF formula is as following:

- We reserve one level in the graph for each clause.
- Each clause starts with a single vertex C_i.
- Before the reserved level, the clause path diverges into m_i branches, one for each literal in the clause.
- At the reserved level, each branch intersects with the literal path of opposite polarity.

- After the reserved level, diverged branches converge into a single vertex again and finally extend to $\tilde{C}_i$.

Figure 5.2 illustrates the conversion from a CNF to a k-level digraph.

Each variable-labeled vertex x_i has exactly two paths to its counterpart $\tilde{x}_i$, the upper solid *true* path and the lower dashed *false* path. We shall also refer to the true path as the path of the positive literal x_i and the false path as the path of the negative literal $\overline{x_i}$. Each clause-labeled vertex C_i has exactly m_i paths to its counterpart $\tilde{C}_i$, where m_i is the number of literals in clause C_i.

For instance, clause $C_1 = x_1 + x_2 + \overline{x_3}$, there are three different paths that connect vertex C_1 to $\tilde{C}_1$: one intersects with the path of the negative literal $\overline{x_1}$, one intersects with the path of the negative literal $\overline{x_2}$ and the last intersects with the path of the positive literal x_3. The claim is that: if there exists an assignment of boolean variables $\{x_1, x_2, x_3\}$ that satisfies all clauses $\{C_1, C_2\}$, then we can find a collection of vertex disjoint paths in the digraph that connect all labeled vertices $\{C_1, C_2, x_1, x_2, x_3\}$ on the left to their corresponding tilde labeled vertices $\{\tilde{C}_1, \tilde{C}_2, \tilde{x_1}, \tilde{x_2}, \tilde{x_3}\}$ on the right, and *vice versa*.

First, suppose the formula is satisfiable. We identify the set of vertex disjoint paths that connect *sans*-tilde vertices to tilde ones as follows: each variable vertex x_i can follow the path of its assigned value to $\tilde{x}_i$. Since clause C_i is satisfied, it must contain some true literal l, whose opposite polarity literal path $\overline{l}$ is available. Therefore, we can pick the literal branch $\overline{l}$ to connect vertex C_i to $\tilde{C}_i$.

Now suppose, we have a set of paths that connect *sans*-tilde vertices to tilde ones. We build a satisfying assignment as follows: Each variable x_i will simply be assigned the value of the path that connects vertex x_i to $\tilde{x_i}$. For clause C_i, there must exist some literal l in C_i, whose opposite polarity literal branch $\overline{l}$ is used to connect vertex C_i to $\tilde{C_i}$ This implies that the literal path l is used to connect its variable vertices since the other path $\overline{l}$ is not available, i.e., literal l is true. Thus clause C_i is satisfied. □

Theorem 5.2. *Optimum scheduling of a traffic matrix on the NoC in the PSM architecture is NP-hard.*

Proof. Figure 5.3 demonstrates a transformation from a k-level digraph to an equivalent PSM digraph. The general steps to construct the PSM graph from a k-level digraph are:

- All vertices from the original digraph are copied into the set of processor vertices.
- For each processor vertex v, we create a corresponding memory vertex v'.
- Each processor vertex and its memory vertex are connected together.
- Each memory vertex v' is connected back to the processor vertices that are supposed to be the neighbors of vertex v in the original digraph. For instance, in Figure 5.1, vertex a has two neighbors c and d, therefore in Figure 5.3, memory vertex a' is connected to processor vertices c and d.

We shall refer to the vertices on the leftmost level in Figure 5.1 as *sources*, and vertices on the rightmost level as *sinks*. Consider a traffic matrix M that encodes the requirement that one packet must be transferred from each source to its paired-up sink. For example, in Figure 5.2, each *sans*-tilde vertex needs to connect to its tilde counterpart.

We consider the following decision problem: given the constructed digraph, is there a schedule implementing all packet transfers in M which completes in $2 \cdot (k-1)$ cycles? Note that each source needs to traverse exactly $2 \cdot (k-1)$ edges to reach a sink; therefore if a schedule meets the bound, no buffering can occur at intermediate vertices. This means that no two packets can share any vertex in their routes, which leads to an instance of DCP, which we showed was NP-hard in Theorem 5.1. Therefore optimum scheduling of M on the constructed PSM graph is NP-hard. □

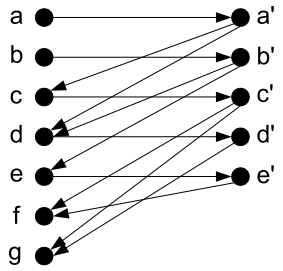

Figure 5.3: A PSM digraph that is equivalent to the 3-level digraph in Figure 5.1.

5.4 Our Scheduling Methodology

We have proved in Section 5.3 that optimum scheduling of a traffic matrix on a NoC in the PSM architecture can not be performed in polynomial time, assuming P $\neq$ NP. Indeed, our proof of NP-hardness used very simple traffic matrices—specifically, the scheduling problem is NP-hard even when the traffic matrix is Boolean valued, and no row or column has more than one entry set to 1.

The intractability of optimum scheduling motivates us to devise a heuristical approach. In this section we will develop such a heuristic to efficiently construct a (possibly suboptimum) schedule for a given interconnection network and traffic matrix.

5.4.1 Congestion

The idea behind our heuristic is to unroll the network K times along the time axis, where K is a customizable parameter. The i-th unrolled network will be called stage i. We shall refer to the vertex in the original digraph as v without superscript. Its unrolled instance in stage i will be referred to as $v^{(i)}$. Each path thus becomes a vertex sequence from its source in its beginning cycle to its destination in its ending cycle in the unrolled digraph. To account for buffering, we need to add a series of *forward* edges that connect $v^{(i)}$ to $v^{(i+1)}$ for all v in the original digraph and all i up till $K-1$. If a path passes an unrolled vertex $v^{(i)}$ through a non-forward edge, vertex v is active for the path during cycle i. Now validity maintenance becomes trivial in the unrolled

digraph: we just need to make sure that no two paths pass the same unrolled vertex with non-forward edges.

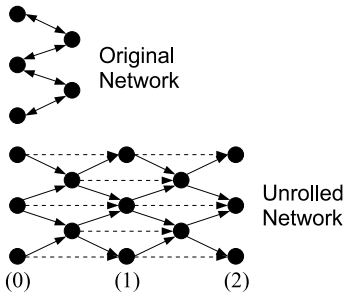

Figure 5.4: A network is unrolled twice. Note that all bidirectional edges become unidirectional because time flows from left to right. Also all forward edges are dashed for differentiation.

To heuristically select a path that will not block others, we develop a notion of *congestion* on edges. The congestion C_e on edge e is defined as:

$$C_e = \sum_{p \in P} \frac{S_p}{d_{e,p}} + \sum_{p \in P} \frac{T_p}{d_{e,p}}$$

where S_p and T_p are the corresponding row and column sums of vertex p in M; the distance $d_{e,w}$ between an edge $e = \{u, v\}$ and a vertex w is $\max\{d(u, w), d(v, w)\}$, where $d(x, y)$ is length of the shortest path between x and y, when all edges are of unit length. The complete heuristic is presented in Algorithm 4.

5.4.2 Heuristic Algorithm

Several key points are:

1. The loop from Line 9 to 12 chooses the path with the most blockage, thereby giving priority to long paths first.

2. The computation of shortest paths in Line 8 with Dijkstra's algorithm is very fast in theory and practice.

3. All vertices in path $p*$ are isolated as in Lines 17 and 20. This guarantees the validity of each *Span*.

4. At Line 25, we always zero out a row with the largest sum in M if its corresponding vertex becomes disconnected from others and repetition of this operation will eventually turn M into an all-zero matrix, guaranteeing finite termination.

5. This procedure is constructive and always produces a valid schedule as long as every non-zero entry in M corresponds to two vertices that are connected in the unrolled digraph.

5.4.3 K and BvN Bound

Algorithm 4 essentially groups many paths together into a K-stage *span*. Hence, it will prompt *infeasible* if there is an entry in the matrix whose source is more than $2K$ edges away from its destination. The total time T in number of cycles is the product of the length of each span and the number of spans needed, assuming all spans are of equal length.

One way to perform scheduling in the PSM architecture is to chose a large enough value for K such that for every packet, a path exists from its source to its destination within a K-stage span. Then the techniques developed by Chang *et al.* [19] for scheduling traffic matrix transfers on a crossbar can

be directly applied; the BvN bound of the traffic matrix (cf. Section 5.2.2.1), determines the number of steps needed. A lower bound on the length of a BvN step in number of cycles is the maximum distance in the PSM graph from processor vertex i to processor vertex j over all i, j such that $M_{i,j} \neq 0$; any step length less than this will preclude some packet being delivered in one step. When applicable, we will compare the number of cycles our schedule takes to the lower bound on the number of cycles taken by the BvN schedule, which is the product of the BvN bound and the lower bound of the length of a BvN step.

5.5 Experimental Results

5.5.1 LDPC Decoding

An LDPC code is a block error correcting code, where there are C bits per block, which include D parity checks [25]. An LDPC code is most naturally represented as a bipartite graph on a set of C code nodes and D check nodes. The decoding algorithm [16] involves iterations of message passing back and forth between connected code and check nodes, and it is this communication that defines the traffic matrix. LDPC codes have very good error correcting capability, but the decoding requires a great deal of computation and communication [17], making LDPC decoding a good candidate for illustrating our work.

We created 6 LDPCs C1–C6 with 40 code and 20 check nodes using randomized code construction techniques [17]. Each entry in the connection

	C1	C2	C3	C4	C5	C6
T	48	34	48	42	38	40
T_B	78	56	52	68	44	66
Reduction	38%	39%	8%	38%	14%	39%

Table 5.1: LDPC: total time T by Algorithm 4 vs. minimum time T_B of BvN decomposition for LDPC codes C1–C6. Both are reported in number of cycles. Reduction in cycle number is reported in the third row.

matrix corresponds to exact one transfer of a result from a code node to a check node or vice versa. The 60 processor nodes are connected to 60 memory nodes by an expander-type interconnection network, where an edge exists between processor i and memory j with probability 0.1—in total, there are 367 edges in the PSM graph.

Computing the schedule on a 1.8 GHz AMD processor took less than 1 minute for each LDPC code. Since Algorithm 4 is executed offline, the performance of our current implementation is acceptable for practical systems.

The total time T in Table 5.1 is obtained with the parameter K that allows us to complete all packet transfers in one span. The results confirm that by unrolling along the time axis, we can effectively time-multiplex vertices: our schedule outperforms the BvN schedule by 30%, on average.

5.5.2 FFT

We studied implementing a 512-point FFT with 256 processing elements (PEs) connected to 64 RAMs. Specifically, PE numbered $(b_8b_7b_6b_5b_4b_3b_2b_1)_2$ is connected to four RAMs numbered $(11b_8b_7b_6b_5)_2$, $(10b_8b_7b_4b_3)_2$, $(01b_6b_5b_2b_1)_2$

	L1	L2	L3	L4	L5	L6	L7	L8
$K = 1$	11	11	10	10	10	10	9	9
$K = 2$	8	9	6	6	5	5	5	5
$K = 3$	5	5	5	5	5	5	5	5
$K = 4$	5	5	5	5	5	5	4	4

Table 5.2: 512-point FFT: Number of spans by Algorithm 4 given level L1–L8 and $K = 1..4$.

and $(00b_4b_3b_2b_1)_2$. Thus we only have $256 \times 4 = 1024$ edges in the $256 + 64$ vertex PSM graph. In this 512-point FFT, excluding the first level where external inputs are read, we pass data through exactly 8 levels of the butterfly. We report the performance of our schedule algorithm for each levels with different values of K.

First, a simple lower bound for the number of spans is $256/(2 \times 16) = 8$: there are 256 packets to exchange at each level, and 2×16 is the limiting bandwidth (since every packet has to use a RAM whose index does not include b_l, where l is the level number). Based on our network connection scheme, 2 groups of 16 RAMs can serve the purpose at each level.

To compare with the lower bound of 8, we need to multiply the number in each row with its K value. And when $K = 1$, schedules generated by our heuristic used merely 25% more spans with just 1024 links. It is very clear in Table 5.2 that a larger K yields a smaller number of spans. But larger K values give larger (worse) products of $K \times$ NumSpan. This is due to our assumption that RAMs are single ported; for multiported RAMs, which allow simultaneous reads from and writes to different addresses, we expect the

performance of larger K scenarios to improve significantly.

5.6 Conclusion

We have developed a general model, complexity results, and heuristics for scheduling static traffic for on-chip networks serving parallel share memories. Our work can be extended in several ways. First, we would like to apply these techniques to dynamic traffic, e.g., by using averaging over windows. Second, we would like to study the *placement* problem, i.e., the problem of mapping computations to processors, so as to get fast schedules. Third we would like to develop predictive models for estimating the true hardware cost of an architecture, including the scheduler overhead.

Algorithm 4 Heuristic to Generate a Schedule

Input: unrolled digraph G and matrix M
Output: Σ—a schedule that completes M

$\Sigma \leftarrow \emptyset$;
while M has positive entries **do**
  Backup M in M';
  $Span \leftarrow \emptyset$;
  Calculate congestion for all edges;
  repeat
    Pick the row in M with the largest sum, record the corresponding vertex as $v^{(0)*}$;
    Compute shortest paths $P = \{p_{v^{(0)*}w^{(K)}}\}$ from $v^{(0)*}$ to all vertices $w^{(K)}$ in the last stage;
    for all $p_{v^{(0)*}w^{(K)}} \in P$ **do**
      Determine the blockage of the path, the sum of congestions on $p_{v^{(0)*}w^{(K)}}$;
      Keep track of the path with the largest blockage in $p*$, which connects $v^{(0)*}$ to $w^{(K)*}$;
    end for
    if the entry in M of $v*$ and $w* > 0$ **then**
      Add the path $p*$ to $Span$;
      for all $v^{(i)}$ in $p*$ **do**
        if $p*$ arrives at $v^{(i)}$ via a non-forward edge **then**
          Remove all edges incident at $v^{(i)}$ in stage i except forward edges;
        end if
        if $p*$ leaves $v^{(i)}$ via a non-forward edge **then**
          Remove all edges incident at $v^{(i)}$ in stage $i+1$ except forward edges;
        end if
      end for
      Decrease the entry in M of $v*$ and $w*$ by one;
    else
      Set the row of $v*$ to zeros;
    end if
  until all entries in M are zeros
  Restore M from M';
  if M is not zero and $Span$ is $\emptyset$ **then**
    Prompt *Infeasible*!
  else
    Add $Span$ to Σ;
    for all P in $Span$ **do**
      Decrease the entry in M of P's first and last vertices by one;
    end for
  end if
end while

Chapter 6

Application II: Scheduling for Reliability

6.1 Background

High defect rates will be a major challenge for future chip designers, both in scaled CMOS as well as emerging nanotechnologies. In a typical Network-on-Chip (NoC) multiple paths exist between a source and a sink. Because of this redundancy, a manufacturing fault on a single interconnect does not necessarily render the resulting integrated circuit useless. In this chapter, we quantify the fault tolerance offered by an NoC. Specifically, we (1.) provide a model for determining the probability that a link in an NoC is faulty, and (2.) measure the impact of link failures on the number of cycles the NoC takes to implement communication.

6.1.1 Defect Densities and Interconnects

As CMOS processes scale, defect densities increase for several reasons, e.g., because of statistical variations in the number of doping atoms in a channel, imperfect planarization of the chip surface between processing steps, the use of subwavelength lithography, etc. Nanotechnologies, such as those based on carbon nanotubes, as well as molecular and quantum devices, suffer from even higher defect rates. Therefore one challenge in future design, be it in

scaled CMOS or an emerging nanotechnology, is providing enough redundancy to get reasonable yields.

The standard approach to coping with high defect rates is to add redundant elements to the design. For example, SRAM caches on modern processors are organized as a matrix of memory cells. The matrix includes extra rows, and if post-manufacturing testing determines that a row contains a defective cell, then the address generation logic is programmed to skip that row.

Redundancy is required not just for devices, but also for interconnects—wires and associated repeater logic [4]. The relative importance of interconnects on the performance of integrated circuits increases with feature size reduction. According to the International Technology Roadmap for Semiconductors, traditional interconnects will be a major bottleneck for technology nodes beyond 45 nm [26].

6.1.2 NoC Overview

An emerging trend for connecting computational elements on a chip is to use a Network-on-Chip [1]. An NoC replaces dedicated point-to-point interconnects with an ensemble of links connected through programmable crosspoints [5]. A key advantage is that the wiring resources can be shared by time-multiplexing and programming the crosspoints appropriately, and so the same communication can be implemented with less interconnect.

In a typical NoC multiple paths exist between a source and a sink, a manufacturing fault on a single interconnect does not necessarily render the

resulting integrated circuit useless. Therefore, an NoC can also help overcome the challenges posed by high defect rates on interconnects.

An NoC can mitigate many other problems associated with conventional interconnects. For example, NoCs are being proposed that offer standard packet-based interfaces to on-chip modules such as processor cores, memories, bus controllers, etc., thereby simplifying design and promoting reuse. Networking techniques that add reliability and Quality-of-Service guarantees have also been proposed. A structured network also means the potential for regular layout, and better circuitry, thus accelerating the physical design flow to achieve timing closure quicker.

6.1.3 Chapter Structure

The rest of this chapter is devoted to quantifying defect rates for interconnects and determining the resilience of NoCs to interconnect defects. In § 6.2 we develop a high-level analytical model for interconnect failure. Specifically, we derive an analytical expression for the probability of a given interconnect failing, and show the model is accurate with respect to a low-level simulation model. In § 6.3 we examine the impact of the interconnect defects on the ability of the NoC as a whole to implement desired functionality. Specifically, we determine how many cycles it takes to perform LDPC decoding on mesh-structured NoCs as a function of the interconnect failure rate.

6.2 Probability of Interconnect Failure

As feature sizes decrease, process variation has emerged as a significant factor that impacts the performance of modern integrated circuits.

Process variation affects the performance of a chip since it causes large variations in delay, power and crosstalk noise. This has direct impact on the yield since the design constraints may not be met for a large number of fabricated chips.

Hardware defects (known as *faults*) in a chip can be caused by a short circuit due to unwanted deposition of metals or other materials, an open circuit due to some missing features, or delay violations. Depending on the nature of the fault, the faults can be of the type *stuck-at-fault, bridging fault, open fault* or *delay fault*. Process variation is a major source of delay faults since it alters the delay values from the designed values causing delay failures.

In this section, we develop analytical models for the probability of interconnect failure due to process variability and verify the models with Monte Carlo simulations. We first qualitatively describe the relationship between process variation and interconnect faults, identifying the key failure mode (§ 6.2.1). Then we use a distributed Elmore delay model and obtain an analytical expression for failure probabilities (§ 6.2.2).

6.2.1 Process Variation and Interconnects

Process variation can be classified into three broad groups [27]: (1.) variation caused by the statistical nature of deposition, (such as variation in the number of dopant atoms the device channel), (2.) variation caused by lithography, specifically photolithography proximity effects, as well as deviations in the mask and lens (which leads, for example, to variation in the length and width of wires, doping regions, poly, etc.), and (3.) variation caused by nonideal chemical-mechanical planarization (such as variation in interlevel dielectric thickness).

A Network-on-Chip is used to provide global communication between functional units, which implies the interconnects it uses are long. This in turn means the NoC interconnects will be built on higher metal layers, which use wires that are significantly wider than those at the lower layers [4]. Faults due to the statistical nature of deposition and due to lithography are more significant on objects which are close to minimum feature size, and are therefore less of a concern for NoC interconnects.

However, variation caused by nonideal chemical-mechanical planarization is significant, specifically, we will see there is significant interconnect delay variation resulting from to multi-conductor pattern erosion and dishing within individual conductors due to chemical-mechanical polishing leads. Nassif [27] has shown that CMP can have a significant impact on conductor thickness, with 3-sigma variations of up to 30% for current process technologies. Conductor line width may also vary, especially for conductors with small line widths

where 3-sigma variation can be up to 20% for interconnects with minimum pitch [27]. For global interconnects, which are typically wide, the effect of dishing will be the major source of variability since dishing effects increases with the width of the interconnect.

Because of the variation in the interconnect dimensions, the parasitic resistance and capacitance will vary from the nominal values causing a shift in the delay from the designed value. With the increasing demand for high performance computing, chips are generally designed with smaller design margins and delay variation beyond the margins will cause chip failure in reality. In NoC systems, the problem of interconnect failure is mitigated by finding alternative paths during the scheduling. For the purpose of robust and reliable scheduling, appropriate models for the probability of interconnect failure are required which we develop in the next section.

6.2.2 Probability of Failure

In NoC applications, the nodes are connected by programmable paths which consist of interconnects interspersed with switching elements in between them. Because of process variations, the delay of the interconnects will be different from the nominal (no variation) values. In a batch of fabricated chips, the delay of each bus will have a probability distribution with the mean typically equal to the nominal value. Therefore, there will be cases when the delay of a interconnect will be greater than the nominal value plus the design margin. In this section, we first compute the probability that a particular

interconnect in the bus fails to meet the delay specification and then find the probability of failure of a bus with a number of interconnects in it, using an RC Elmore delay model.

The surroundings of a long global bus will typically vary along the length of the bus giving rise to variation of different degrees along the length. In our analysis we therefore consider the case of length independent (uniform) variation as well as the length dependent variation where we consider a piecewise uniform model for the variation.

The Elmore delay is a widely accepted model for general RC tree delay. Elmore delay has a high degree of fidelity since an optimal or near optimal design obtained by using an Elmore delay will also produce a near optimal design based on more accurate delay models [28]. For a distributed, uniform interconnect modeled as an RC network, the Elmore delay τ is given by: $\tau = k \cdot r \cdot c \cdot l^2$, where k is a constant, 0.38 for this case, r and c are the resistance and capacitance per unit length and l is the length of the interconnect. In order to compute the worst case delay for a signal line, the effective capacitance value for the signal line is given by $c = c_g + 2c_c$, where c_g is the self capacitance per unit length and c_c is the coupling capacitance per unit length to the nearest neighbor [29]. For the outermost lines in a bus, c will be given by $c_g + c_c$ since it is coupled to only one neighbor.

For nonuniform variation across the length of a bus, the interconnect is modeled as piecewise uniform segments, as shown in Figure 6.1. Because of this variation along the length, the interconnect can be modeled as shown

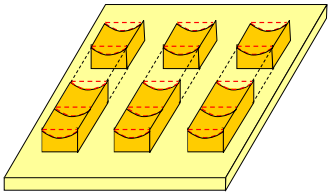

Figure 6.1: Dishing of piecewise uniform interconnects.

in Figure 6.2, where the interconnect is divided into N piecewise uniform segments. The interconnect delay is computed by the Elmore approximation as follows:

$$\begin{aligned} \tau &= k \cdot \big[(C_1 + C_2 + \ldots + C_N) \cdot R_1 + (C_2 + C_3 + \ldots \\ &\quad + C_N) \cdot R_2 + \ldots + C_N \cdot R_N\big] \end{aligned} \tag{6.1}$$

where k is a constant, R and C are the resistance and capacitance terms for each segment, respectively.

In the presence of variation due to dishing of the interconnect lines, as depicted in Figure 6.1, the resistance of the interconnects will vary, whereas the capacitance values will not be affected much since the height of the sidewall practically stays unaffected. In the remainder of this section, we determine the probability of delay failure due to process variation for both the uniform and piecewise uniform buses.

6.2.3 Uniform Interconnects

For modeling the effect of process variation, the resistance of the interconnect needs to be considered as a Gaussian random variable r with mean

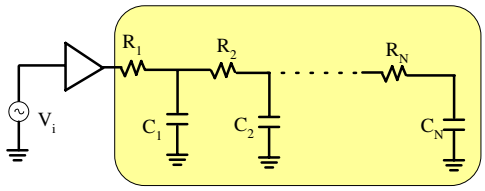

Figure 6.2: RC model for piecewise uniform interconnect lines.

m_r and variance σ_r. The probability that the delay of the interconnect will be greater than a given value D, represented as $P[\tau > D]$ is:

$$\begin{aligned} P[\tau > D] &= P[k \cdot r \cdot c \cdot l^2 > D] = 1 - P\left[r < \frac{D}{k \cdot c \cdot l^2}\right] \\ &= 1 - F(\frac{\frac{D}{k \cdot c \cdot l^2} - m_r}{\sigma_r}) \\ &= Q(\frac{\frac{D}{k \cdot c \cdot l^2} - m_r}{\sigma_r}) \end{aligned} \tag{6.2}$$

where r is the resistance per unit length, c is the capacitance per unit length, l is the length of the interconnect and F is the cumulative distribution function of the resistance. Here Q is the classical error function, and we use Borjesson's approximation [30] to evaluate it as follows: $Q(x) \approx \frac{e^{-x^2/2}}{\sqrt{2\pi}[(1-a)x + a\sqrt{x^2+b}]}$, where $a = 1/\pi$ and $b = 2\pi$

6.2.4 Piecewise Uniform Interconnects

In case of a piecewise uniform interconnect with N segments, resistance of each segment will have different mean m_i and variance σ_i in general. Considering a Gaussian distribution for resistance, the overall mean and standard deviation of the Elmore delay can be found as the mean and standard deviation of a linear combination of Gaussian variables [31]. For the N-segment

piecewise uniform interconnect, the final expressions for mean and standard deviation are given by: $m_\tau = \sum_{i=1}^{N} m_i \sum_{j=1}^{N} C_j$ and ${\sigma_\tau}^2 = \sum_{i=1}^{N} {\sigma_i}^2 \sum_{j=1}^{N} {C_j}^2$. Finally, using the mean and standard deviation of the Elmore delay, the probability of the delay being greater than a given value is expressed as before:

$$P[\tau > D] = 1 - P[\tau < D] = Q(\frac{D - m_\tau}{\sigma_\tau}) \tag{6.3}$$

6.2.5 Analytical Model

For performance reasons, links in an NoC are made out of sets of interconnects, running in parallel, rather than a single interconnect. Using the expressions of delay failure for individual interconnects derived above, the probability of failure for a interconnect with n interconnect lines (essentially, an n-bit bus) can be calculated. As mentioned before, the worst case delay for the two outermost lines in the bus will be different from the delay of the remaining $n-2$ lines and the probability of delay failure will also be different. Let the probability of delay failure be P_{fout} for the two outermost lines and P_f for the remaining lines. If the variation of each interconnect in the bus are independent, then the overall probability of failure in the bus will be $P_{Fail} = 1 - (1 - P_{fout})^2(1 - P_f)^{n-2}$. However, the assumption of independent variation will make the probability of interconnect failure highly pessimistic. In the real case, the variations in different interconnects of a bus are never uncorrelated since a high degree of spatial correlation exists between these interconnects. Therefore, completely correlated variation will be more realistic for this case, which leads to the following expression for the probability of

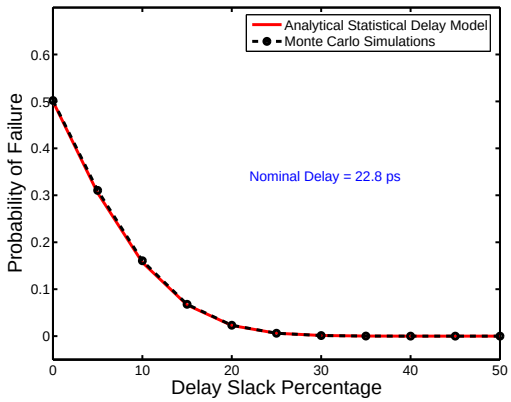

Figure 6.3: Probability of failure as a function of percentage delay slack for a typical global interconnect in 90nm technology. The probability of failure obtained from the analytical expressions closely matches the probability computed from Monte Carlo simulations.

interconnect failure: $P_{Fail} = max(P_{fout}, P_f)$. Since typically the delay of the outer lines will be less than that of the other lines, the delay slack for the outer lines will be more and P_{fout} will be less than P_f. This reduces $max(P_{fout}, P_f)$ to P_f in the above equation.

6.2.6 Comparison with Simulation Results

For verifying the correctness of the probability model proposed above, we chose a typical global interconnect from the 90nm technology node. The 3σ variation of the dishing height was chosen to be 30% of the nominal value. The corresponding delay variation using a Gaussian distribution for the dishing was computed using the above-mentioned method. Monte Carlo simulations were also performed for the Elmore delay variation. The probability of failure

for different percentage delay slacks is shown in Figure 6.3. It can be easily noted that the probability values computed using our analytical expressions closely match the values obtained through Monte Carlo simulations validating the correctness of our method.

6.3 Impact of Interconnect Failure

In this section, we study the effects of interconnect failure on the ability of the NoC to implement desired communication. Our assumption is that interconnect failures are identified by post-manufacturing test, and that the routes for the NoC are determined based on the test results and the traffic derived from the algorithm to implement.

The benefits of an NoC are most pronounced in settings where there is a large amount of communication. Many DSP subsystems have this property. These systems are typically implemented as an ensemble of processing elements (PEs) operating in parallel, with a very large amount of communication between PEs. In our experiments we will focus on an NoC organized as a mesh, implementing the communication for Low-Density Parity Check (LDPC) decoding, which is computationally very challenging, and requires a great deal of communication [17].

6.3.1 Experiments

In earlier work [32], we proved that computing optimum schedules is NP-hard and developed heuristics for computing schedules.

We apply the same heuristics to compute schedules for mesh-structured NoCs with defective interconnects implementing LDPC decoding. We assume interconnects are defective independently of each other, with a fixed probability p, and report the average number of cycles of the schedule as a function of p.

An LDPC code is a block code, where there are C bits per block, which include D parity checks. It is most naturally represented as a bipartite graph on a set of C code nodes and D check nodes. The decoding algorithm [16] involves iterations of message passing back and forth between connected code and check nodes, and it is this communication that defines the traffic matrix.

In Figure 6.4 we plot the average number of cycles in the schedules computed by our heuristic as a function of the defect probability p. The NoC is organized as a 23×23 mesh, and the traffic matrix corresponds to the communication pattern for an LDPC code with 48 check nodes and 96 code nodes. Specifically, for each p, we created 50 NoCs with that defect probability. The average is taken only over instances in which we were able to generate a schedule. As p increased, we began to see more instances where our heuristic could not produce any schedule, e.g., because the NoC became disconnected. This trend is shown in Figure 6.5.

6.4 Conclusion

In conclusion, we identified delay faults caused by chemical-mechanical polishing as being the primary source of NoC interconnect failures, and de-

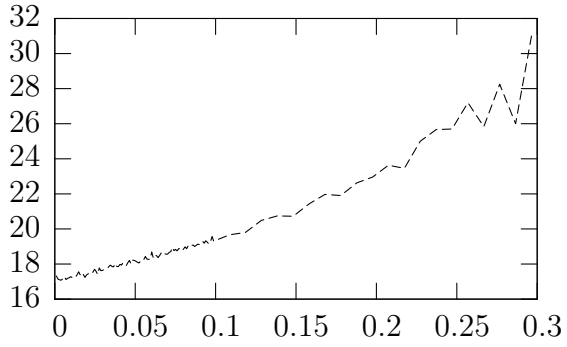

Figure 6.4: Number of cycles in schedule vs. link failure probability

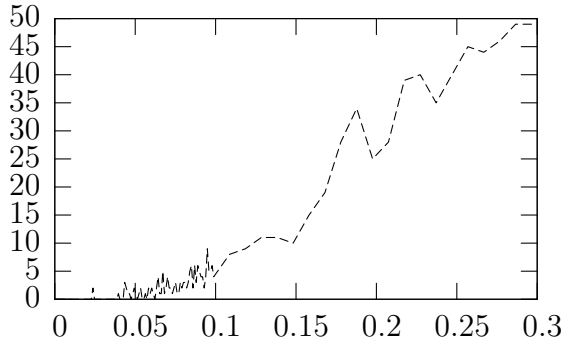

Figure 6.5: Number of infeasible instances vs. link failure probability

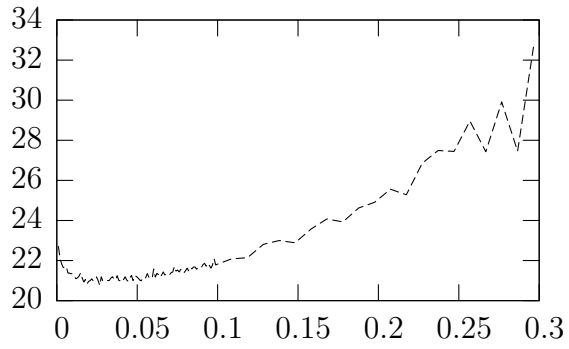

Figure 6.6: Normalized total delay vs. link failure probability

rived an analytical formula approximating the probability of a link failure as a function of cycle time. Monte Carlo simulations against a detailed model demonstrated that our formula is a good approximation. We demonstrated for LDPC decoding computations how the number of cycles required to implement the communication varied with the link failure probability.

These two results are connected—the probability of a link being faulty increases with the clock frequency, and the number of cycles it takes to implement communication increases with the link defect rate. Consequently, it is natural to ask for what clock frequency (equivalently, what cycle time), the *total* time to implement the communication is minimized. We plot the total time to implement the computation as a function of link failure probability in Figure 6.6, and see that the optimum is achieved for a link failure probability around 0.04.

Chapter 7

Summary of Contributions

7.1 Summary

We have developed general models and algorithms for implementing algorithms with on-chip networks. The combinatorial problems of binding computations to process elements and scheduling on-chip networks are both proved to be NP-hard. We first analyze the pseudo-polynomial time algorithm for cycle-free topologies and then derive efficient heuristics for unbuffered networks. Evaluations with LDPC decoding and FFT show that our algorithm can approach theoretical bounds fairly close with even sparse networks. We further apply our results to buffered networks, and formulate an efficient heuristic based on network unrolling. Lastly, we turn our focus to post silicon reprogramming of networks. And simulation based on our statistical model tells us that by rescheduling the network after manufacture, we can actually improve performance and yield simultaneously.

7.2 Relationship to Prior Work

7.2.1 Key Differences

Our treatment of the network scheduling problem differs from multiprocessor routing [10] because our algorithm solves *general* traffic matrices, whereas classic work almost entirely focuses on routing individual *permutation* matrices. Our work differs from the routing problem considered in physical design [33] in the following way: the main objective in physical design is to provide connectivity, while minimizing combinations of area/power/delay; we to increase the utilization of interconnects by time-multiplexing them. In our work on the PSM architecture, packets are allowed to be buffered at intermediate stages—a packet does not have to be delivered through the interconnection network in a single cycle. This expands the applicability of our theory and algorithm to a much larger spectrum of applications.

For majority of the on-chip network research that will be described in detail in the next subsection, researchers consider design constraints and objective of their networks in terms of *statistical metrics*. These researchers perform scheduling dynamically—routers are responsible for forwarding packets or setting up circuits based on incoming traffic at the instance of time. The assumption is that the network's behavior hinges on the input data, which is the best we can obtain for general purpose computation and networking applications. But for our applications, on-chip networks quite often operate independently of the actual incoming data. Thus, there is no need for the overhead of dynamic scheduling, since access patterns are known a priori. As

a result, the amount of data and time to transfer them are given to designers exactly. Inspired by BvN decomposition [19], we solve the problem in the context of a determined future.

Lastly, considering the implementation circuitry, statically scheduled routers are much simpler than dynamic counterparts. Dynamic routing requires the ability to buffer data in the presence of conflicts; these buffers are quite expensive. In our approach, we may buffer the packet, but we can tailor the buffer sizes with different schedules for different performance envelops. We also need to store the schedule in a ROM, but this is relatively much smaller than SRAM/flop based buffers needed for resolving conflicts in a dynamic router.

7.2.2 Detailed Literature Review

7.2.2.1 Bus-based Networks

Commercial bus-based networks such as *AMBA* [34] and *CoreConnect* [35] have been widely recognized. Sonics *MicroNetwork* [36] is an early attempt that automatically generates the communication subsystem in a highly customizable SoC design flow. The network can be abstracted as a TDMA bus, and handles physical issues inside its root and agent transceivers. *STbus* [37] provides similar features, but it also supports more advanced topologies such as partial or full crossbars. *Lotterybus* [38] addresses the problem from a statistical point of view, offering low latency for high priority burst traffic and effective bandwidth guarantee.

Although shared buses are easy to design and interface with, they are criticized in [1, 5, 39] for their poor scalability. A point-to-point network with switchers or routers controlling the packet flow is envisioned by Dally *et al.* [1, 39], along with a light weight communication protocol that enhances the performance and reliability. Bjerregaard *et al.* [40] summarizes the cutting edge research in a detailed survey.

7.2.2.2 Network Architectures

Historically, high speed interconnects were extensively studied for building multiprocessor machines. Classics such as [10] provide comprehensive coverage of network architectures and routing algorithms.

The *Maia* system [41] leverages heterogeneous function blocks to achieve high performance and power efficiency for DSP applications. The communication is serviced by a hierarchical mesh network based on circuit switching. And the network itself is designed ad hoc according to the specific implementation of the algorithm. Maia exploits the traffic patterns in DSP algorithms for partition of function blocks and choosing the network architecture, which is well resonated in our work. However our approach differs in that: (1.) we focus on a general topology packet switching network connecting a sea of homogeneous processing elements, (2.) we provide theoretical analysis and practical heuristics for near optimum schedule given the static traffic and (3.) last but not the least, our flow is completely independent of the specific algorithm to implement, i.e., orthogonal to the design of computation blocks [42].

To facilitate packet routing and physical design process, most network designers adopt regular structures, such as meshes or trees. The *RAW* processor first appeared in [43] is composed of tiled cores. In its latest incarnation, the "scalar operand network" [44] includes two statically scheduled meshes and one dynamically scheduled mesh. Instructions to fetch data from adjacent tiles are directly exposed into ISA. The compiler is responsible for inserting communication instructions and optimizing the binary program for mesh networks. One key observation is that the latency of the network can be amortized because of dominating locality in general purpose computing. Compared with our flow, developers of RAW heavily focus on data and computation partition to minimize traffic, which can be considered as a much larger superset of our placement technique. Our effort is to shorten the time of communication given a batch of transfers whereas in [43], an average time cost is assumed for each communication instruction. Another recent implementation of the operand network can be found in [45] as part of the *TRIPS* project.

NOSTRUM [46, 47] is an NoC system built upon a medium sized mesh and a layered protocol stack. The combined backbone based application specific platform is then delivered to SoC designers for further mapping. An interesting point made by NOSTRUM creators is that for multimedia devices and consumer electronics, traffic exhibits high locality, therefore higher order topologies are rarely necessary and meshes provide suitable level of redundancy for robust operation. The *Scalable Programmable Integrated Network (SPIN)* described in [48] selects fat-tree as its topology and features a carefully de-

signed router optimized for small packets with efficient buffer management. Wiklund *et al.* in [49] employs a two dimensional mesh network *SoCBUS* and packet connected circuit (PCC), i.e., the packet locks the path that it traverses in the network. This path setup technique helps achieve appealing bandwidth with low latency in the dynamically scheduled network. An application to an Internet core router design is also presented in [50]. Other structures such as octagon [51] and star connect [52] have also been proposed.

Rijpkema *et al.* [53] explore the trade off between guaranteed and best effort services in on-chip router design. An on-line matrix scheduling scheme is applied in the input-queued architecture. Our work also enjoys the concise formulation of traffic matrices, but embraces an off-line scheduling scheme without intermediate buffering.

Lately, on-chip networks with specially designed protocols are proposed to support memory coherency [54, 55], one of the key challenges in micro architecture.

7.2.2.3 Synthesis and Design Support

Project *PROTEO* [56] provides a library of parameterized components aiming at decoupling logic design from underneath technology. Complex network designs can be synthesized from components in the library, whose parameters can be further tuned to support target process. Pinto *et al.* considers the general communication system synthesis problem in [57]. A latency insensitive network design is further proposed in [58], which focuses on a high

performance on-chip interconnect for IP blocks with minimal impact to the back-end design flow. Zhu *et al.* [59] presents a hierarchical approach for modeling networks based on a class library. An irregular network generation procedure is described in [60]; both temporal and spacial information are exploited in the optimization process. Memory optimization issues in networking chips are explored in [61].

The deep combinatorial problem of assigning cores to network nodes is discussed in [62, 63]. A complete synthesis flow of NoCs for multiprocessor SoC appears in [64], which includes *xpipes* [65] a library of soft macros, *xpipesCompiler* [66] that generates network components based on xpipes, and *SUNMAP* [67] that maps cores onto the selected network architecture.

DSPIN [68] presents detailed physical implementation of an extended version of the SPIN NoC. Gilabert *et al.* [69] explores high-dimensional topologies with transaction level simulation under physical design constraints. NoC designs are also subjected to process and temperature variations even during early stages, as discussed in [70].

Appendices

Appendix A

Complexity Analysis

In this chapter, we analyze the complexity of the general network scheduling problem and its variations.

A.1 Scheduling General Networks

A.1.1 BvN Decomposition for Crossbar

We begin with this very important special case. If the graph representation of the network is a complete bipartite graph that connects sources to sinks, Chang *et al.* [19] have shown how to efficiently compute an *optimum* schedule for completing the traffic matrix, where optimum means the least number of cycles. In design terms, this implies that a crossbar structure is used to connect all parts that need to communicate with each other together. Given n sources and n sinks connected via a crossbar, and an $n \times n$ traffic matrix M, their approach is built on a result due to Birkhoff and von Neumann on the decomposition of matrices, which in turn is based on the existence of perfect matches in a Δ-regular bipartite graph [71]. We will refer to the schedule computed by Chang *et al.* [19] as the *BvN schedule* of M. Their schedule takes exactly $\max\{\max_i \sum_j M_{ij}, \max_j \sum_i M_{ij}\}$ cycles; we will refer

to this value as the *BvN bound* of M. Note that the BvN bound is tight as long as every source or sink can only engage in one packet transfer during a single cycle.

A.1.2 Scheduling for Sparse Networks

When the network is not as dense as a crossbar, the scheduling problem becomes much harder. Based on definitions from Section 2.4, we can easily establish following result:

Lemma A.1. *Given a graph G and a traffic matrix M, determining whether M is G-feasible is NP-hard.*

This is due to the fact that a feasible matrix can be one-to-one mapped to a vertex disjoint path set (VDPS) in the graph. And the vertex disjoint path problem is known to be NP-hard from the book [14][Problem ND40]. This implies that determining whether a traffic matrix can be discharged in *one* cycle with a given graph is NP-hard. Therefore, finding a minimum cycle schedule for the traffic matrix is at least NP-hard.

Theorem A.1. *Given a graph G and a traffic matrix M, finding the optimum schedule to discharge the matrix is NP-hard.*

A.2 Variations

A.2.1 Tree Topology

When the network topology is cycle-free, i.e., for every pair of vertices, there exists a unique path that connects them together, a fast algorithm is available to compute the schedule.

Theorem A.2. *When there is no cycle in the graph G, an optimum schedule to discharge a traffic matrix M can be found in pseudo-polynomial time.*

The algorithm and proof are provided in Section 3.2.

A.2.1.1 Vertices with Arbitrary Capacities

It is noteworthy that if we allow multiple paths to pass through a vertex in one cycle, this problem becomes NP-hard again. More precisely, besides the graph G and matrix M, we are also given a vertex capacity function $\phi : V \mapsto Z^+$. And during each cycle, we allow up to $\phi(v)$ paths to pass through vertex v in the graph. Thus far, we are focused on the scenario in which $\phi(v) = 1$ for all v in the graph. When ϕ can take values greater than 1, we are no longer constrained to generate only VDPS anymore. We can thus redefine following notion: a matrix M is (G, ϕ)-feasible iff there exists a set of paths satisfying: 1) there are exactly M_{ij} paths that connect vertex i to vertex j and 2) for any vertex v in the graph, there are no more than $\phi(v)$ paths in the set that pass through v. Accordingly, we need to generate the least number of (G, ϕ)-feasible matrices to form a minimum schedule. And we shall prove the follow theorem:

Theorem A.3. *Given a cycle-free graph G, a traffic matrix M and a vertex capacity function ϕ defined above, if the function ϕ is allowed to take arbitrary integer values, finding an optimum schedule to discharge the matrix is NP-hard.*

Proof. We will reduce this problem to the edge coloring problem. Suppose an arbitrary simple graph $C = (V_C, E_C)$ is to be edge-colored, where $V_C = \{v_1, \cdots, v_n\}$, we can construct a graph $G = (V_G, E_G)$, a traffic matrix M and a capacity function ϕ such that:

1. $V_G = \{b_1, \cdots, b_n, l_1, \cdots, l_n\}$, we shall refer to b_i as branch vertices and l_i as leaf vertices;

2. $E_G = \{\{b_i, b_{i+1}\}\}_{i=1..n-1} \cup \{\{b_i, l_i\}\}_{i=1..n}$; note that E_G is cycle free and the only path that connects vertex l_i to l_j is by the edges in $\{\{b_i, l_i\}\} \cup \{\{b_i, b_{i+1}\}, \cdots, \{b_{j-1}, b_j\}\} \cup \{\{b_j, l_j\}\}$ assuming $i \leq j$; Figure A.1 illustrates the construction of graph G;

3. M is a $2n \times 2n$ Boolean traffic matrix, where the only ones are of the following: $M(l_i, l_j) = 1$ iff $\{v_i, v_j\} \in E_C$, i.e., we transfer a packet from l_i to l_j iff v_i and v_j is connected by a direct edge in graph C;

4. $\phi(b_i) = n$ for branch vertices and $\phi(l_i) = 1$ for leaf vertices, for all i between 1 and n.

We first show that if k colors are sufficient to edge-color graph C, then there is a k cycle schedule that discharges the matrix M subjected to the

constraints of the topology G and the capacity function ϕ constructed above. Given an edge coloring of graph C, consider c, one of the k colors, we can extract a set of edges $E_c = \{\{v_i, v_j\}$ that is colored with $c\}$. A $2n \times 2n$ Boolean traffic matrix m_c can be constructed such that $m_c(l_i, l_j) = 1$ iff $\{v_i, v_j\} \in E_c$. We shall prove that m_c is (G, ϕ)-feasible.

To prove the feasibility, we just need to construct a set of paths that satisfies the constraints out of G and ϕ. As explained above, to connect l_i to l_j, we can take the path $\{\{b_i, l_i\}\} \cup \{\{b_i, b_{i+1}\}, \cdots, \{b_{j-1}, b_j\}\} \cup \{\{b_j, l_j\}\}$ if $i \leq j$ or $\{\{b_i, l_i\}\} \cup \{\{b_i, b_{i-1}\}, \cdots, \{b_{j+1}, b_j\}\} \cup \{\{b_j, l_j\}\}$ if $i > j$. And it is clear that by following these paths, we can discharge all packets encoded for the edges in E_c. The only missing step is to show that these paths won't violate the constraints imposed by the capacity function ϕ. To see why this is the case, we need to first point out that $|E_c| <= n/2$ because no two edges in E_c share a vertex as they are all of the same color. This instantly guarantees that we won't violate the capacity constraint at branch vertices since they all have capacity $\phi(b_i) = n \geq n/2$. And since no two edges in E_c share a vertex, this tells us that no leaf vertex is involved in more than one packet encoded from E_c, i.e., each leaf vertex has at most one path that passes through itself. Hence the capacity constraint $\phi(l_i) \leq 1$ is not violated either. Based on what we have shown, we can essentially map the set of edges of the same color into a (G, ϕ)-feasible matrix. Since we can color the graph with k colors, we can find k (G, ϕ)-feasible matrices that sum up to the total traffic matrix M due to the fact $E_C = \cup_c E_c$. This completes a valid k cycle schedule to discharge

M given the graph and vertex capacities.

Now we move to the second part: if there is a k cycle schedule that discharges the matrix M subjected to the constraints of the topology G and the capacity function ϕ constructed above, then the original graph C is k-colorable. Suppose we have a k cycle schedule for M, this implies that $M = \sum_{c=1..k} m_c$, where each matrix m_c is (G, ϕ)-feasible. Since the only non-zero entries in M are at locations $M(l_i, l_j) = 1$, we know that the non-zero entries in m_c are also at locations (l_i, l_j) for some i and j and they can only be 1. Now we can derive a coloring for graph C as following: we assign a unique color c for each matrix m_c, and if $m_c(l_i, l_j) = 1$, we color the edge $\{v_i, v_j\}$ with color c and we shall prove that this is indeed a valid edge coloring. Suppose two edges $\{v_i, v_j\} \in E_C$ and $\{v_i, v_k\} \in E_C$ share the vertex v_i, we need to prove that they will be not assigned the same color with the way we color the graph. Notice that $\phi(l_i) = 1$, this guarantees that for any m_c, there is only one entry that can involve the vertex l_i, otherwise the capacity constraint will be violated. Hence, edges $\{v_i, v_j\}$ and $\{v_i, v_k\}$ must be encoded in two different matrices, and thus they must be assigned different colors. □

A.2.2 Dependencies

Another interesting variation of this scheduling problem has to do with dependency among packets to transfer. First off, data and control dependency is built in many real world applications. For instance, an MPEG decoder must rely on previously decoded frames to restore the current one as specified by

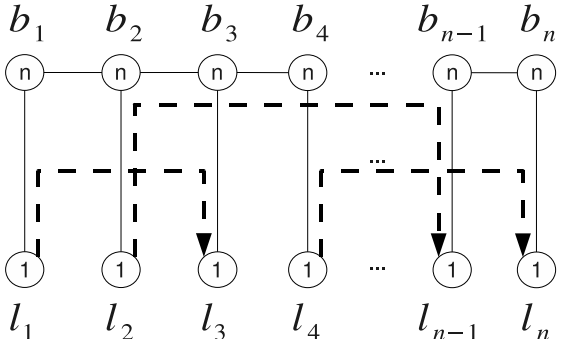

Figure A.1: Branch and leaf vertices and their connections. Each circle denotes a vertex and its label is the value of the capacity function ϕ. Notice that three packets are marked in the diagram with dashed arrows. And they can all complete in the same cycle due to the newly introduced capacity constraint.

the standard. Second, when we implement an algorithm, we need to introduce some artificial dependency. When we time multiplex a processing unit or the on-chip storage is limited, operations and data must be carefully pipelined to avoid conflict. This translates into a weak ordering among packets that we need to pass among on-chip blocks. It is well known that the multi-machine scheduling problem subjected to precedence constraints [14][Problem SS9] is NP-hard. Therefore, even with very simple topologies, precedence constraints will render the network scheduling problem NP-hard.

A.2.3 Network Design

There is a related design problem that's equally appealing. Given communication requirements encoded in a traffic matrix M and certain constraints

on the communication network, it is natural to ask what is the best topology to connect vertices together. Here the best topology is the one that yields the best possible schedule in terms of transfer cycles. To begin with, let's assume all n vertices are fully connected with the least number—$(n-1)$ edges. And further simplification can be made by demanding that no vertex has more than two edges attached. This effectively reduces our design problem to seeking an optimum linear order that minimizes that maximum cross bandwidth, where bandwidth is defined as the number of packets that cross a vertex. The minimum cut linear arrangement problem [14][Problem GT44] tells us that finding such an optimum linear order is NP-hard even when the traffic matrix is Boolean.

A.3 Approximation

A great deal of effort has been put into solving route selection and packet scheduling problems in the context of high performance interconnect. The route selection step deals with selecting a set of paths for a set of given packets in order to optimize one or multiple objectives subjected to a suite of constraints. During the packet scheduling step, it is assumed that the route for every packet is given and the sole issue is to program routers and time multiplex links to implement those routes subjected to graph topology and hardware capacity. Our flow iteratively generates schedulable routes and reduces the traffic, which solves these two problems simultaneously. But historically, they are treated separately.

The objectives of the route selection step often include minimizing the total or the maximum length of all packet routes and minimizing congestion among routes, which we shall see later are highly related to the packet scheduling step. The object can also be minimizing the weighted cost of all routes when the network is organized in hierarchy and inter-domain transfers are more expensive than intra-domain ones. In reality, we quite often optimize a combination of two or more objectives mentioned above and keep unused ones in bound as our constraints. The most popular objectives of packet scheduling include minimizing the last packet's arrival time, i.e., *makespan* and maximizing weighted success rate of packet delivery. Constraints can be a delay requirement on each individual packet and internal buffer sizes, which tell us how many packets can be stored at an internal node at any point of time. Both problems in their most general forms are proved to be NP-hard and approximation algorithms have flourished in research.

Leighton *et al.* proved in [72] that given the specified routes of a collection of packets, there always exists an optimal schedule that completes all transfer with $O(C + D)$ steps. The number C is the congestion among the routes, which is the maximum number of routes that pass through a single edge, while D is the maximum length of all routes. The proof was based on Lovàsz Local Lemma and thus did not explicitly construct the desired schedule. Note that $C + D$ is within a constant factor to the optimum, therefore the route selection problem can be almost independently solved when proper objectives and constraints are applied. A series of work focused on centralized

and distributed scheduling algorithms has been in publication.

Bertisimas *et al.* [73] provided an asymptotically optimal approximation algorithm that combined the route selection and packet scheduling together. It was based on fluid relaxation, i.e., multi-commodity flow solutions. We can always cast a path connection problem into a flow problem by throwing away the integer requirements. And we can always round down the fractional flow at each edge to an integer to make it a feasible solution for path connection. However, due to the continuity of flows, a number of packets will not be transfered after we schedule vertices based on the rounded down integer solution. Bertisimas *et al.* argued that when the number of packets was dominating the graph size, this directly rounded down solution from fractional flows was actually asymptotically optimal. More precisely, suppose the solution for multi-commodity flow problem is C^*, their algorithm takes $C^* + O(\sqrt{C^*})$ time and C^* is an absolute lower bound since it is a relaxation of the original problem. However, for this scheme to work, the lower bound must be much larger than the number of edges in the graph, i.e., $C^* \gg |E|$. This assumption is rarely valid with on-chip network. But when the pathological scenario is encountered, this rounding technique does provide the best solution.

For general path selection problems, [74] surveys a comprehensive set of variations and techniques. A polynomial time approximation scheme can be found in [75]. An interesting analysis of many to one routing on a grid appears in [76], in which a solution within $O(\log D)$ factor of the optimum is attained. A local control based randomized packet scheduling algorithm that

approaches $O(C + D + \log^{1+\epsilon} N)$ is presented in [77], where N is the network size.

Bibliography

[1] W. Dally and B. Towles, "Route Packets Not Wires: On chip Interconnection Networks," in *Design Automation Conference*, Jun. 2001.

[2] M. Horowitz, R. Ho, and K. Mai, "The Future of Wires," Invited workshop paper for SRC conference, 1999. [Online]. Available: citeseer.ist.psu.edu/horowitz99future.html

[3] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli, "Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design," in *Design Automation Conference*, 2001.

[4] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 2005.

[5] J. Turner and N. Yamanaka, "Architectural Choices in Large Scale ATM Switches," *IEICE Transactions*, 1998.

[6] E. Lee and D. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Trans. Comput.*, vol. 36, no. 1, pp. 24–35, 1987.

[7] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonte, J.-H. Ahn, N. Jayasena, U. J. Kapasi, A. Das, J. Gummaraju, and I. Buck, "Merrimac: Supercomputing with Streams," in *ACM/IEEE Conference on Supercomputing*, 2003.

[8] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: A Many-Core x86 Architecture for Visual Computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–15, 2008.

[9] K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. John-Wiley, 1999.

[10] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan-Kaufmann, 1991.

[11] J. Cong and Z. Pan, "Interconnect Performance Estimation Models for Design Planning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 6, pp. 739–752, June 2001.

[12] A. Abou-Seido, B. Nowak, and C. Chu, "Fitted Elmore Delay: A Simple and Accurate Interconnect Delay Model," *IEEE Transactions on VLSI Systems*, vol. 12, no. 7, pp. 691–696, July 2004.

[13] R. Li, D. Zhou, J. Liu, and X. Zeng, "Power-Optimal Simultaneous Buffer Insertion/Sizing and Uniform Wire Sizing for Single Long Wires," in *Pro-*

ceedings of the IEEE International Symposium on Circuits and Systems, May 2005, pp. 113–116.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability.* W. H. Freeman and Co., 1979.

[15] K. Maharatna, E. Grass, and U. Jagdhold, "A 64-point Fourier Transform Chip for High-Speed Wireless LAN Application Using OFDM," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, Mar. 2004.

[16] R. G. Gallager, "Low-Density Parity-Check Codes," Ph.D. dissertation, MIT, Cambridge, MA, 1962.

[17] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 404–412, March 2002.

[18] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms.* MIT Press, 1989.

[19] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load balanced Birkhoff-von Neumann Switches, part I: One-Stage Buffering," *Computer Communications*, 2001.

[20] N. H. E. Weste and K. Eshragian, *Principles of CMOS VLSI Design.* Addison-Wesley, 1993.

[21] T. Johnson and U. Nawathe, "An 8-core, 64-thread, 64-bit Power Efficient SPARC SoC (Niagara2)," in *International Symposium on Physical Design*, 2007.

[22] Juniper Networks, "High Speed Switching Device," US Patent 5,905,725, 1999.

[23] S. Iyer, R. Zhang, and N. McKeown, "Routers with a Single Stage of Buffering," in *ACM SIGCOMM*, 2002.

[24] A. Prakash, S. Sharif, and A. Aziz, "An $O(\lg^2 n)$ algorithm for output queuing," in *IEEE Infocom*, 2002.

[25] T. Cover and J. Thomas, *Elements of Information Theory.* Wiley Interscience, 1991.

[26] SIA, International Technology Roadmap for Semiconductors, 2005.

[27] S. Nassif, "Modeling and Analysis of Manufacturing Variations," *Proceedings of the Custom Integrated Circuits Conference*, May 2001.

[28] R. Gupta, B. Krauter, B. Tutuianu, J. Willis, and L. T. Pileggi, "The Elmore Delay as Bound for RC Trees with Generalized Input Signals," in *Design Automation Conference*, 1995, pp. 364–369.

[29] J. Qian, S. Pullela, and L. Pillage, "Modeling the "Effective capacitance" for the RC interconnect of CMOS gates," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Dec. 1994.

[30] P. Borjesson and C. Sundberg, "Simple Approximation of the Error Function $Q(x)$ for Communications Applications," *IEEE Transactions on Communication*, Mar. 1979.

[31] H. Stark and J. Woods, *Probability and Random Processes with Applications to Signal Processing.* Prentice Hall, 2002.

[32] X. Wu, A. Prakash, M. Mohiyuddin, and A. Aziz, "Scheduling Traffic Matrices on General Switch Fabrics," in *Hot Interconnects*, Stanford University, CA, Aug. 2006.

[33] N. Sherwani, *Algorithms for VLSI Physical Design Automation.* Springer, 2005.

[34] ARM Ltd, "AMBA Bus Specification from www.arm.com."

[35] IBM Ltd, "CoreConnect Bus Architecture www.ibm.com/chips/products/coreconnect."

[36] D. Wingard, "MicroNetwork-Based Integration for SoCs," in *Design Automation Conference*, 2001.

[37] S. Murali and G. D. Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation," in *Design Automation and Test in Europe Conference*, 2005.

[38] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs," in *Design Automation Conference*, 2001.

[39] L. Benini and G. DeMicheli, "Networks on Chips: a New Paradigm for Component-Based MPSoC Design," 2002. [Online]. Available: citeseer.ist.psu.edu/benini04networks.html

[40] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1, 2006.

[41] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. Rabaey, "A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, 2000.

[42] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2000. [Online]. Available: citeseer.ist.psu.edu/keutzer00system.html

[43] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring It All to Software: Raw Machines," *IEEE Computer*, vol. 30, no. 9, pp. 86–93, 1997.

[44] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalable Operand Network: Design, Implementation and Analysis," MIT-LCS-TM-644, Technical Report, MIT, 2004. [Online]. Available: http://www.lcs.mit.edu/publications/specpub.php?id=1757

[45] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. G. McDonald, S. W. Keckler, and D. Burger, "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," in *ACM/IEEE Symposium on Networks-on-Chip*, Princeton University, NJ, May 2007.

[46] S. Kumar, A. Jantsch, and J. P. Soininen, "A Network on Chip Architecture and Design Methodology," in *International Conference on VLSI*, Apr. 2002, pp. 105–112.

[47] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "The NOSTRUM Backbone - a Communication Protocol Stack for Networks on Chip," in *Proceedings of VLSI Design, India*, 2004. [Online]. Available: citeseer.ist.psu.edu/millberg04nostrum.html

[48] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet Switched Interconnections," in *Design Automation and Test in Europe Conference*, 2000. [Online]. Available: citeseer.ist.psu.edu/guerrier00generic.html

[49] D. Wiklund and D. Liu, "SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, 2003.

[50] ——, "Design of an Internet Core Router Using the SoCBUS Network on Chip," in *Proceedings of IEEE International Symposium on Signals, Circuits and Systems*, 2005.

[51] F. Karim, A. Nguyen, S. Dey, and R. Rao, "On-chip Communication Architecture for OC-768 Network Processors," in *Design Automation Conference*, 2001.

[52] S. J. Lee, S. J. Song, K. Lee, J. H. Woo, S. E. Kim, B. G. Nam, and H. J. Yoo, "An 800MHz Star-Connected On-Chip Network for Application to Systems on a Chip," in *Proceedings of IEEE International Conference on Solid-State Circuits, Digest of Technical Papers*, 2005.

[53] E. Rijpkema, K. Goossens, A. adulescu, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort services for Networks on Chip," in *Design Automation and Test in Europe Conference*, 2003. [Online]. Available: citeseer.ist.psu.edu/rijpkema03trade.html

[54] N. E. Jerger, L. shiuan Peh, and M. Lipasti, "Circuit-Switched Coherence," in *ACM/IEEE Symposium on Networks-on-Chip*, Newcastle University, UK, Apr. 2008.

[55] E. Bolotin, Z. Guz, I. Cidon, R. Ginosar, and A. Kolodny, "The Power of Priority: NoC Based Distributed Cache Coherency," in *ACM/IEEE Symposium on Networks-on-Chip*, Princeton University, NJ, May 2007.

[56] I. Saastamoinen, D. Siguenza-Tortosa, and J. Nurmi, "Interconnect IP Node for Future System-on-Chip Designs," in *Proceedings of IEEE International Workshop on Electronic Design, Test and Applications*, 2002.

[57] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, "Constraint-Driven Communication Synthesis," Technical Report, UC Berkeley, 2002. [Online]. Available: citeseer.ist.psu.edu/pinto02constraintdriven.html

[58] L. Carloni and A. Sangiovanni-Vincentelli, "Coping with Latency in SoC Design," *IEEE Micro*, vol. 22, no. 5, pp. 24–35, Sep. 2002.

[59] X. Zhu and S. Malik, "A Hierarchical Modeling Framework for On-Chip Communication Architectures," in *International Conference on Computer-Aided Design*, 2002.

[60] W. Ho and T. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 2, pp. 174–190, Feb. 2006.

[61] D. Whelihan and H. Schmit, "Memory Optimization in Single Chip Network Switch Fabrics," in *Design Automation Conference*, 2002.

[62] 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*, 2003.

[63] S. Murali and G. D. Micheli, "Bandwidth Constrained Mapping of Cores onto NoC Architectures," in *Design Automation and Test in Europe Conference*, 2004.

[64] D. Bertozzi, A. Jalabert, S. Murali, R. T, S. Stergiou, L. Benini, and G. D. Micheli, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, Feb. 2005.

[65] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs," in *International Conference on Computer Design*, 2003.

[66] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli, "xpipesCompiler: a Tool for Instantiating Application Specific Networks on Chip," in *Design Automation and Test in Europe Conference*, 2004.

[67] S. Murali and G. D. Micheli, "SUNMAP: a Tool for Automatic Topology Selection and Generation for NoCs," in *Design Automation Conference*, 2004.

[68] I. Miro-Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture," in *ACM/IEEE Symposium on Networks-on-Chip*, Newcastle University, UK, Apr. 2008.

[69] F. Gilabert, S. Medardoni, D. Bertozzi, L. Benini, M. E. Gomez, P. Lopez, and J. Duato, "Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework," in *ACM/IEEE Symposium on Networks-on-Chip*, Newcastle University, UK, Apr. 2008.

[70] B. Li, L.-S. Peh, and P. Patra, "Impact of Process and Temperature Variations on Network-on-Chip Design Exploration," in *ACM/IEEE Symposium on Networks-on-Chip*, Newcastle University, UK, Apr. 2008.

[71] J. van Lint and R. Wilson, *A Course in Combinatorics*. Cambridge University Press, 1992.

[72] F. T. Leighton, B. M. Maggs, and S. B. Rao, "Packet Routing and Job-Shop Scheduling in O(congestion + dialation) Steps," in *ACM Symposium on the Theory of Computing*, 1988.

[73] D. Bertsimas and D. Gamarnik, "Asymptotically Optimal Algorithms for Job Shop Scheduling and Packet Routing," *Journal of Algorithms*, vol. 33, no. 2, pp. 296–318, 1999.

[74] S. Plotkin, "Competitive Routing of Virtual Circuits in ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1128–1136, 1995.

[75] K. Jansen, R. Solis-Oba, and M. Sviridenko, "Makespan Minimization in Job Shops: a Polynomial Time Approximation Scheme," in *ACM Symposium on the Theory of Computing*, 1999, pp. 394–399.

[76] Y. Mansour and B. Patt-Shamir, "Many-To-One Packet Routing on Grids," in *ACM Symposium on the Theory of Computing*, 1995, pp. 258–267.

[77] R. Ostrovsky and Y. Rabani, "Universal O(congestion + dilation + log1+ϵN) Local Control Packet Switching Algorithms," in *ACM Symposium on the Theory of Computing*, 1997, pp. 644–653.

Vita

Xiang Wu was born on Apr 2, 1979 in China. His hometown is a small city by the YangTze River (also known as Chang Jiang) named Shashi, which literally means "the city of sand" in Chinese. He received his B.S. in physics from Tsinghua University in Beijing in the year of 1999. His thesis was about surface plasmon resonance and its application in sensoring. He received his M.S. in Industrial Engineering from the University of Arizona in Tucson in the year of 2003. He worked under Prof. Julia L. Higle in the Systems and Industrial Engineering department on modeling and optimizing vehicle reservation systems. He is currently a Ph.D. candidate in the Electrical and Computer Engineering department of the University of Texas at Austin. He is working under Prof. Adnan Aziz on a variety of topics, including on-chip network scheduling, hardware and software verification techniques and high performance massive scale software systems. He has published 6 research papers since 2003 when he arrived in Austin.

His professional career centers around verifying SoC designs. In the summer of 2004, he worked at Jasper Design Automation, Mountain View CA on speeding up formal verification engines. From Jan, 2005 to Feb 2006, he worked at Britestream Networks, Austin TX on testing DRAM controllers and on-chip switch fabrics. From Apr 2006 to present, he is working at Advanced

Micro Devices, Austin TX on a series of projects of building low power mobile processors.

Permanent address: 10017 Hidden Falls Dr.
Pearland, TX 77584

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.