BALANCING PERFORMANCE, AREA, AND POWER IN AN ON-CHIP NETWORK

by

Brian Gold

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in a partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Computer Engineering

July, 2003
Blacksburg, Virginia

Keywords: Router, crossbar switch, network, virtual channels, power, area, system on chip, message passing, single chip computer, SCMP

BALANCING PERFORMANCE, AREA, AND POWER IN AN ON-CHIP NETWORK

Brian Gold

James M. Baker, PhD, committee chair
Department of Electrical and Computer Engineering

Abstract

Several trends can be observed in modern microprocessor design. Architectures have become increasingly complex while design time continues to dwindle. As feature sizes shrink, wire resistance and delay increase, limiting architects from scaling designs centered around a single thread of execution. Where previous decades have focused on exploiting instruction-level parallelism, emerging applications such as streaming media and on-line transaction processing have shown greater thread-level parallelism. Finally, the increasing gap between processor and off-chip memory speeds has constrained performance of memory-intensive applications.

The Single-Chip Message Passing (SCMP) parallel computer sits at the confluence of these trends. SCMP is a tiled architecture consisting of numerous thread-parallel processor and memory nodes connected through a structured interconnection network. Using an interconnection network removes global, ad-hoc wiring that limits scalability and introduces design complexity. However, routing data through general purpose interconnection networks can come at the cost of dedicated bandwidth, longer latency, increased area, and higher power consumption. Understanding the impact architectural decisions have on cost and performance will aid in the eventual adoption of general purpose interconnects.

This thesis covers the design and analysis of the on-chip network and its integration with the SCMP system. The result of these efforts is a framework for analyzing on-chip interconnection networks that considers network performance, circuit area, and power consumption.

This work has been generously supported by the National Science Foundation through a Graduate Research Fellowship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

What will the digital system of 2010 look like? What design trends make the current architectures of microprocessors impractical for a billion transistor chip? What possible solutions exist to the design challenges facing engineers in the next few years? Can architects improve the testability, time-to-market, and yield of future designs?

These questions form the motivation for the material presented in this thesis. Where current designs use global, ad-hoc wiring structures, the time has come to consider general purpose networks to connect on-chip subsystems. In the first chapter, current trends in microprocessor design are reviewed and motivation for on-chip networking is presented. The tradeoffs in using a general-purpose interconnect must be understood, so a framework is presented for analyzing the cost and benefit of various architectural design decisions.

1.1 Recent Trends

The well-recognized International Technology Roadmap for Semiconductors (ITRS) provides an analysis and projection of near- and far-term semiconductor fabrication capabilities. Based on the latest available roadmap [ITRS02], high-performance microprocessors are expected to contain up to one billion transistors by the year 2007 and over four billion transistors by 2013.

While it may be technically feasible for a design to contain a billion transistors or more, today's design paradigms will not translate into successful billion-transistor architectures. To accommodate so many gates on a single chip, fabrication technologies continue to shrink. As feature sizes become smaller, the cross-sectional area of wires decreases, causing wire resistance to increase and signal delays to grow. In a billion-transistor architecture, signals will not reach across the chip within one or two clock cycles. In fact, estimates show that less than 1% of a chip will be reachable in a single clock cycle [Agarwal00].

Despite increasing wire delays, microprocessor designers have continued to integrate new features into the existing paradigm of single-threaded execution. For the past thirty years, the focus has been on increasing the number of instructions that can be simultaneously executed from a single instruction stream. While exploiting this instruction-level parallelism (ILP) has netted performance improvements, the size of the monolithic microprocessor has grown tremendously.

This growth has several impacts, most notably increasing area, power consumption, and design cost. Increasing the area of a design results in fewer chips per wafer, and hence a higher fabrication cost. A larger chip also has a higher probability of defects, and thus a lower yield. Structures designed to mitigate the impact of defects are costly to implement since they may require design changes in several chip subsystems, each change being unique to that component.

Power consumption is rapidly becoming a focus for engineers throughout the design cycle. Architects, circuit designers, and layout engineers must be cognizant of the impact their decisions have on power. Power dissipation impacts every application and target market: handheld devices, desktop workstations, and large-scale servers can all benefit from improved power-design methodologies. While improvements can be made at the gate and transistor level, architects must consider power consumption in order to achieve dramatic benefits.

Finally, the continued growth in the complexity of single-threaded architectures has led to increased design and testing costs. The number of engineers required to design, validate, and test a modern microprocessor is constantly growing, while increased competition has resulted in shorter design cycles and lower profit margins.

While area, power, and complexity have suffered with increasing architectural features to exploit ILP, the performance gains have continually diminished. With applications continuing to demand performance improvements, architects have begun to look at thread-level parallelism (TLP). Where ILP

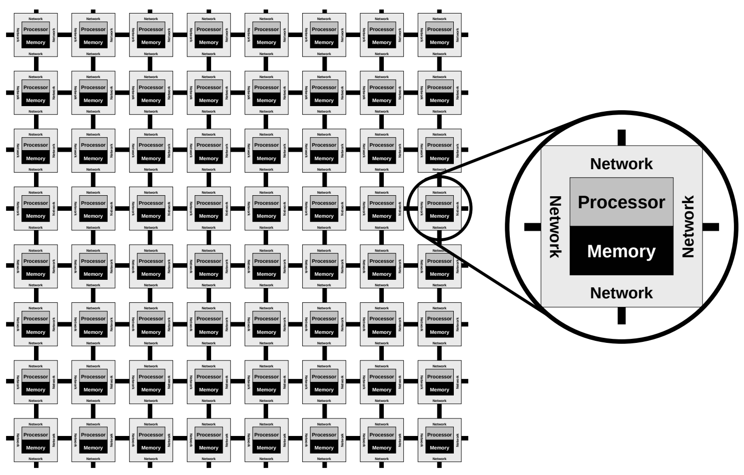


Figure 1.1: The SCMP Parallel Computer consists of interconnected tiles that contain processing, memory, and network components.

exploits the most fine-grained parallelism available, recent studies indicate many applications may contain more parallelism at the thread level [Diefendorff97]. Emerging architectures such as chip multiprocessors (CMPs) and simultaneous multithreading (SMT) are designed to exploit both TLP and ILP. However, CMP and SMT architectures do not generally address the previously mentioned area and design cost issues, and they maintain a shared link to external memory, failing to address wire latency problems.

1.2 The SCMP Parallel Computer

The single-chip message-passing (SCMP) parallel computer is designed to exploit thread-level parallelism using localized, structured wiring to reduce design and testing costs and improve power consumption [Baker02]. The SCMP design integrates several processor-memory tiles on a single chip and connects the tiles with a general-purpose interconnect. A schematic picture of the SCMP design is shown in Figure 1.1.

Each tile in the SCMP system is identical, consisting of a processor, memory, and networking components (see Figure 1.2). The processor is a 32-bit RISC core with added features for managing multiple threads in hardware and sending network messages. It is estimated that in 5-10 years an SCMP chip can be built with 16-64 processors and 4-8MB of memory per processor [Baker02].

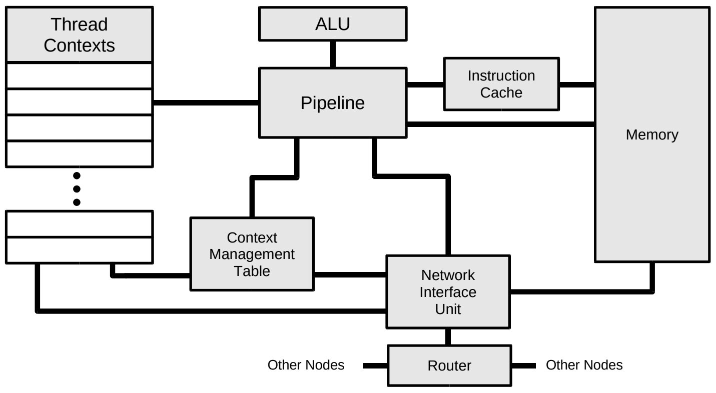


Figure 1.2: A single SCMP node

The entire SCMP system has been designed to be scalable with respect to wire latency. Nodes are arranged in a point-to-point network with nearest-neighbor connections to keep wire lengths at a minimum. Where most shared-memory systems are not scalable beyond a few nodes (snooping bus systems) or have global wiring requirements (directory systems), the SCMP system uses message passing to communicate data between nodes [Culler98]. By integrating memory and CPU in each tile, no wires are longer than those connected to the neighboring tiles.

1.3 On-Chip Networking

Consider the design of a complex integrated system today. Any large chip is going to be composed of several subsystems, and each of those subsystems is likely made up of a number of components. Typically, information is shared between components and subsystems through dedicated wiring. If two components are at opposite ends of the chip, wires may very well stretch across the entire design to accommodate the interconnection. As an example of the impact of the resulting wire delays, consider the extremely deep pipeline of the Pentium IV, where two stages are reserved for signal propagation alone [Hinton01].

The global wiring used today is considered ad-hoc because the signal lines are typically placed as needed and each design will be different. As new features or subsystems are added, wiring needs will likely change. Ad-hoc wiring is costly in terms of design time and testability. Fabrication defects that occur in signal paths can be difficult to design around since each connection path is designed uniquely. Redundant wiring would be costly in terms of area and design time.

Designing a high-speed digital system becomes difficult if the entire chip is tied to a master clock. Clock skew effectively prevents cross-chip components from being synchronized to the same clock, and various structures and techniques have been developed to synchronize subsystems that exist in multiple clock domains [Dally98]. With ad-hoc wiring, implementing clock synchronization increases design time and complexity.

An on-chip network removes the global, ad-hoc nature of subsystem interconnections. Rather, information is passed in the form of messages between nodes in the network. Nodes can be connected through short, localized paths to reduce the impact of wire delays. The wiring between nodes is highly structured and redundancy can be built in with little cost in area or design time. Each node can exist in a local clock domain, and the network can accommodate the necessary synchronization circuitry [Dally01].

Using on-chip networking can come at the cost of dedicated bandwidth, longer latency, increased area, and higher power consumption. Not all systems will be candidates for using a general-purpose interconnect, despite the potential benefits. Quantifying the costs and benefits associated with an on-chip network will aid engineers looking to move towards a structured design.

1.4 Thesis Overview

This thesis will focus on the design issues involved in using a general-purpose interconnect in the SCMP architecture. Data is passed between tiles using the network, while components within each tile use conventional dedicated wiring to carry data and control signals. While researchers have begun to extol the benefits of using a general-purpose interconnect in chip design [Dally01], more analysis is required before industry adoption can begin. This thesis will discuss design issues in using an on-chip network in the SCMP architecture and present a framework for quantifying the costs and benefits associated with using such an interconnect.

The analysis consists of two components: analytical models and simulation-based observations. Analytical models will quantify the delay, area, and power of a pipelined router implementation. Simulations consist of application benchmarks run in an instruction set simulator and synthetic network traffic used in an RTL-level network simulation. The optimum network architecture will be determined by balancing performance, area, and power estimates.

The remainder of this thesis is organized as follows. In the next chapter, relevant background information covering interconnection networks is given. This background chapter will define basic terms and give an overview of the networking taxonomy. In Chapter 3, related research is presented, including parallel computer networks, on-chip networking, and quantifying cost and performance in a network. The fourth chapter gives a detailed description of the design and implementation of the router and network interface of the SCMP system. The next chapter gives an analysis of the router implementation, including analytical expressions for delay, area, and power, and simulation results of performance. The simulations cover both synthetic network traffic and realistic application performance. Based on the results and observations in Chapter 5, the final chapter presents conclusions and future directions for this research.

Chapter 2

Background

Using an on-chip network means that data is passed between subsystems in the form of messages. This thesis concerns direct networks where the components (processors, memories, etc.) are connected through point-to-point links. Messages travel through the network by making one or more hops between the source and destination tiles.

In this chapter, a general overview of interconnection network terminology and concepts is given. Important design parameters will be discussed including network topology, routing and switching algorithms, and flow control techniques. In characterizing an interconnection network, it is important to understand the performance metrics commonly used. An overview of the SCMP network will be presented from the perspective of an application programmer or processor microarchitect.

2.1 Network Performance Metrics

The time between when a message is sent and the complete message arrives at the destination is referred to as *latency*. For this thesis, latency will be measured in two ways. The time between when the message header enters the network and the end of the message is ejected from the network is called *network latency*. When software benchmarks are used to provide performance measures under typical application loads, latency will include the time for pointer arithmetic, memory accesses, etc. as messages are built. This form of latency measure that includes software overhead is called *system latency*. Both measures are important, and will be used throughout the analysis.

Often used in conjunction with latency is the notion of a network's *throughput*. Throughput is the maximum traffic a network can accept per unit time, typically measured as bytes or packets per node per cycle. Throughput is commonly measured from a Burton Normal Form (BNF) plot of latency versus accepted traffic, both functions of offered traffic [Duato02]. The throughput corresponds to the maximum accepted traffic rate where latency approaches infinity, as shown in Figure 2.1.

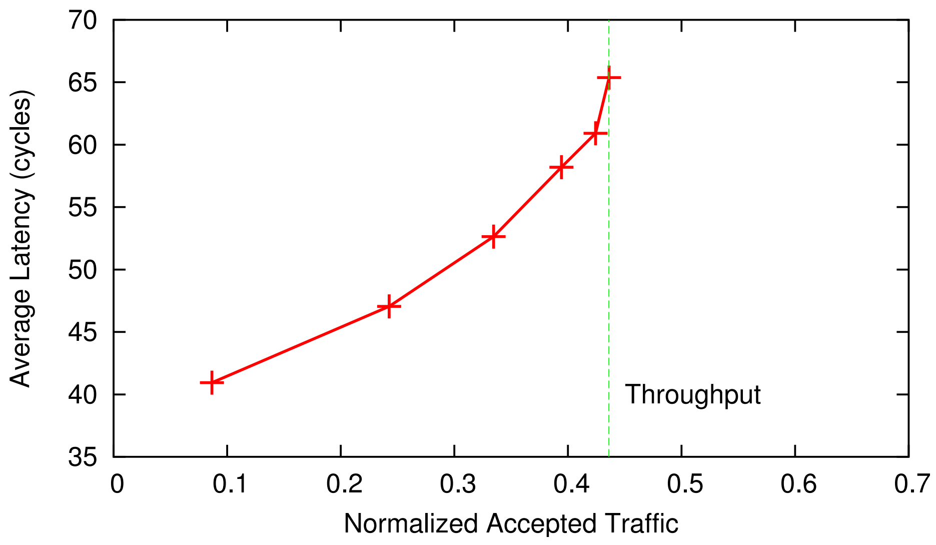


Figure 2.1: A BNF plot illustrating throughput as the maximum accepted traffic.

Another commonly used metric is *bandwidth*. In the context of this thesis, bandwidth will be used to characterize the performance capabilities of a given network structure, or topology (see Section 2.2). Among the numerous bandwidth measures, most important in this thesis is the bisection bandwidth, measured as the minimum number of wires connecting two equal partitions of nodes [Duato02]. The bisection bandwidth will be used to characterize the network activity when traffic is uniformly distributed across the nodes, as in the case of synthetic network benchmarks.

When measuring latency and throughput, there are several ways to quantify and report the results. One technique is to use software application benchmarks that generate network traffic in a practical, realistic context. Software benchmarks include latency introduced by the processor pipeline, e.g. due to pointer arithmetic and memory access. Alternatively, synthetic network benchmarks can be used to generate network traffic with some desired statistical properties in an attempt to stress a network design in a generic way.

Synthetic benchmarks also enable the testing of network components alone, isolated from software-generated overhead.

2.2 Interconnection Network Taxonomy

The nature of an on-chip network can be classified according to its topology, routing protocol, switching mechanisms, and flow control techniques. This section presents the network design alternatives that are relevant to on-chip networking, and the SCMP network in particular.

2.2.1 Network Topology

The topology of a network is defined by the shape and structure of the interconnected nodes. Popular choices include members of the k-ary n-cube family, including 2-D meshes, rings, tori, hypercubes, and Omega networks [Hwang97]. Other possible topologies include fat trees, cube-connected cycles, and star graphs, among others. Figure 2.2 shows several possible topologies for a network.

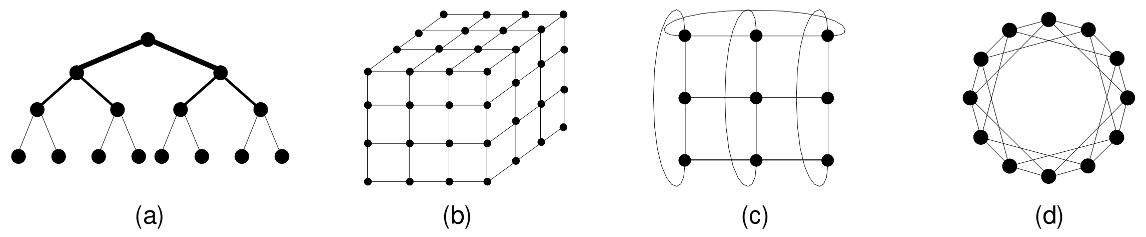


Figure 2.2: Several common network topologies: (a) fat tree, (b) 4-ary 3-cube, (c) torus, (d) ring.

While interconnection networks in conventional multiprocessors frequently take the form of higher-dimensional topologies [Hwang97], on-chip networks will likely use networks such as a 2-D mesh or folded torus with lower dimensionality to keep wire lengths short. Figure 2.3 illustrates the 2-D mesh and folded torus topologies.

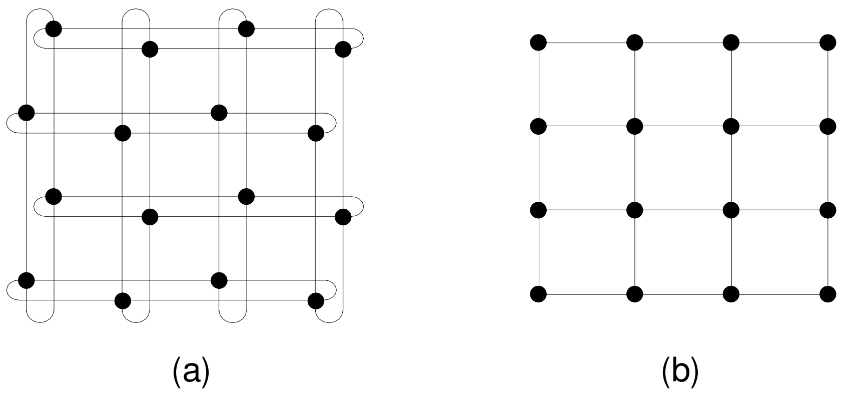


Figure 2.3: Two-dimensional topologies: (a) folded torus and (b) 2-D mesh.

2.2.2 Routing Algorithms

The path a message takes through the network is determined by the routing protocol. Routing can be either deterministic, adaptive, or a combination of both. In a deterministic routing algorithm, a message's path is known precisely from the source and destination nodes, while an adaptive routing algorithm will decide the path based on current network conditions. Deterministic routing algorithms generally offer simpler implementations, while adaptive algorithms can provide better performance and fault tolerance.

Some networks use a hybrid system that reserves some channels for deterministic routing and others for adaptive paths, combining the benefits of the two alternatives at the cost of additional area, wiring, and power [Mukherjee02].

The simplest deterministic algorithm is dimension-order routing, where messages are transmitted fully in each dimension, beginning with the lowest dimension available. For example, in a 2-D mesh network, a message is transmitted first along the "x" dimension until reaching the column containing the destination node. Then, the message moves along the "y" dimension until the destination is reached. An analogous algorithm can be applied to networks with higher dimensions.

Adaptive algorithms are used to improve performance in the presence of localized traffic or to provide fault-tolerance in the network. Several subclassifications of adaptive routing can be made. Among them, minimality and progressiveness are important to consider in on-chip network applications. An algorithm which always moves the message closer to the destination node is termed a *minimal* routing algorithm; otherwise, the protocol allows *misrouting*. A *progressive* algorithm only moves the message forward, while a *backtracking* algorithm can reverse direction and choose an alternative route that differs from the previously chosen path [Duato02].

Deadlock

In addition to its simplicity, another advantage of dimension-order routing is deadlock avoidance. Deadlock occurs when messages are unable to be transmitted due to a circular dependence on resources. The messages cannot proceed because each one is waiting for another node which is also blocked.

Dimension-order routing avoids deadlock by requiring that circular dependencies, or cycles, cannot occur. Other routing algorithms can avoid deadlock by increasing the number of resources available in the network. One way to guarantee deadlock-free operation in adaptive routing networks is to add channels that are routed in a deadlock-free, deterministic fashion. If messages cannot continue on the adaptive channels, they may be transmitted through the deterministic routes to clear the deadlock.

Originally developed to guarantee deadlock-free routing in wormhole networks (see section 2.2.3), *virtual channels* are multiplexed on top of a single physical channel to give the appearance of a larger number of resources. Virtual channels, although originally developed for deadlock avoidance, can also be used to improve flow control performance, and will be discussed in that context shortly.

2.2.3 Switching Mechanisms

Switching techniques control how and when message data is connected from input to output channels. Switching operation can be defined by how messages are broken up and how the subsequent pieces are transmitted between nodes. Many of the popular switching techniques used in parallel computers were originally developed for local- and wide-area networks. Others have been developed specifically with high-performance computing in mind.

In circuit switching, a message's route is secured before the message itself enters the network. Typically, this is achieved by transmitting a header probe from the source to the destination. As the probe travels in the network, it reserves resources along the way. An acknowledgement is sent back to the source to confirm that the circuit has been set up, and the message can then be transmitted. When long messages are present, especially continuous data streams, circuit switching can be advantageous. On the other hand, the latency incurred in sending the header probe and corresponding acknowledgement can make circuit switching prohibitive for short, frequent messages.

Another commonly used alternative is packet switching, where messages are broken into fixed size pieces, termed packets. Each packet is routed independently, and therefore must contain enough control and routing information to be delivered. Packets are also fully buffered at each node in the path, which is why packet switching is also commonly referred to as store-and-forward switching. When messages are short and frequent, packet switching can most effectively use network resources.

Frequently, packet widths exceed available physical channel widths, meaning packets must be transmitted in pieces over several cycles. In virtual cut-through (VCT) switching, packet headers are forwarded as they are received, rather than waiting for the entire packet to arrive. In the absence of blocking traffic, VCT switching has a lower latency than packet switching. VCT switching requires the same available buffer space as packet switching so that, in the case when a packet header is blocked, the entire packet can be stored.

To lessen the buffering requirements of VCT switching, wormhole switching breaks a message into smaller flow-control digits, or *flits*. Only the header flit(s) contains routing information, reducing the overhead incurred with packet or VCT switching. In the presence of blocking traffic, flits can be distributed along the path in the network. Wormhole switching requires only enough buffer space to store a few flits in each router, so the overall storage requirements are less than those in packet or VCT switching [Dally90].

In conventional multi-chip interconnection networks, physical channel widths are typically limited due to small pin counts. Flits are often broken into several *phits* that fit into the physical channel width. With on-chip networks, however, physical channel widths can be significantly larger, but storage requirements and their associated power and area consumption may become more restricted than in multi-chip networks.

2.2.4 Flow Control Techniques

Data is synchronized between network nodes through flow control techniques. Flow control mechanisms allocate and release buffering space and pass credits to inform neighboring nodes of resource availability. In circuit switching, for example, the header probe and acknowledgement packet are examples of flow control. In packet, VCT, or wormhole switching, flow control circuitry keeps track of available buffer space on neighboring nodes, and allocates the space required to transmit a given packet or flit.

One common implementation is to use a source-controlled system, where all state information is kept at the sending end of a transmission link. The sender has information on how many buffer slots are available, which channels are allocated, etc. Credits are passed across a channel from receiver to transmitter to indicate when new space is made available or when a link becomes free.

Virtual Channel Flow Control

Virtual channels, also called virtual lanes, were originally developed to avoid deadlock, as discussed above. However, Dally [Dally92] showed that using virtual channels in a flow control paradigm could reduce the latency of message transmission in a wormhole network. Without virtual channel flow control, a message receives exclusive access to output channels. When a message is blocked, however, the channels in the message's path are now unavailable to other messages that could otherwise proceed. By multiplexing the channel access, virtual channel flow control allows the physical channel to be used by flits that continue to progress in the network. Figure 2.4 illustrates the virtual channel principle.

Virtual channel flow control isn't free, of course. Besides the increased storage required to buffer several messages simultaneously, the flow control circuitry must also be increased to pass information regarding which virtual channel is being used. Credits passed from receiver to transmitter must indicate the applicable virtual channel, so physical channel widths are wider than flow control without virtual channels.

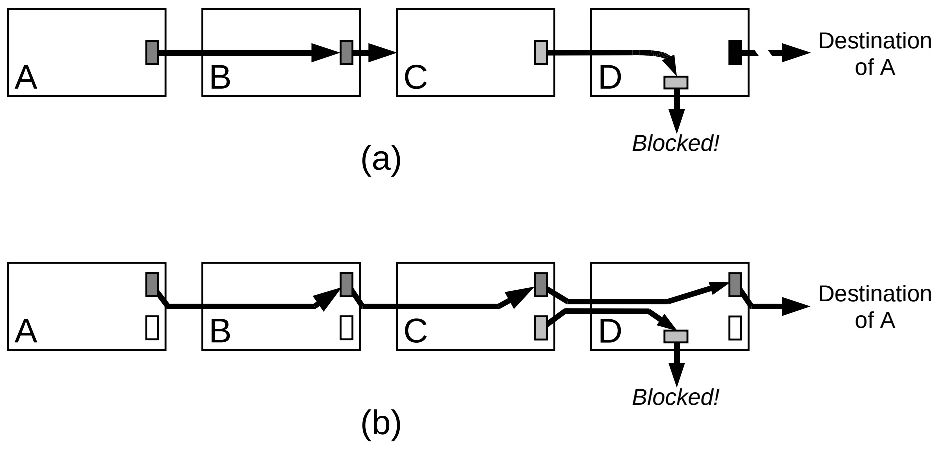


Figure 2.4: Virtual channel flow control. In (a), without virtual channels, the message from A is blocked when the message from C is blocked. In (b), two virtual channels mean the message from A still gets through to its destination. The physical channels have not increased; rather, the virtual channels are multiplexed across each physical channel.

2.3 The SCMP Network

The design of an interconnection network is often dictated by the components being connected, and the SCMP network is no different. The SCMP processing tiles are designed to operate on instruction streams with the network operations controlled through a subset of the SCMP instruction set. Table 2.3 lists the SCMP network instructions, their operands, and the desired result.

The SCMP system supports thread and data messages. Thread messages spawn a new thread context on another tile, while data messages transfer blocks of local memory to a different node. A thread message consists of the starting instruction pointer (IP) and initial context register values. When a thread message is received, the Network Interface Unit (NIU) obtains a context slot in the Context Management Table (CMT) and stores the IP and context registers in the local node.

Data messages consist of a base memory address, an address stride, and one or more data words. Data messages can be constructed at the sender by trans-

Table 2.1: Network-related operations in the SCMP instruction set.

Instruction	Operands	Description
SEND	rs1	Send 1 data word
SEND2	rs1, rs2	Send 2 data words
SEND2E	rs1, rs2	Send 2 data words and end message
SENDE	reg	Send 1 data word and end message
SENDH	reg, type, imm	Send msg. header, imm. operand
SENDH	reg1, type, reg2 [,imm]	Send msg. header, reg. operand
SENDM	reg1, reg2	Send data words from memory
SENDME	reg1, reg2	Send data words and end message

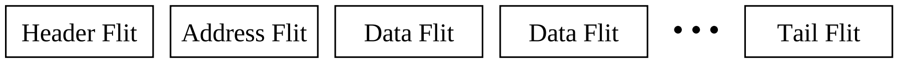


Figure 2.5: SCMP message format.

mitting register values (using SEND or SEND2 instructions) and/or using the block transfer instruction SENDM. The SENDM instruction sets up a direct memory transfer by specifying the starting address, local stride, and number of words to transfer. The NIU initiates the memory requests and builds the message directly from memory, freeing the processor to continue operating. The NIU memory access is given a lower priority than the pipeline memory requests.

Either message type ends when a SENDE, SEND2E, or SENDME instruction is executed. These opcodes are identical to the SEND, SEND2, and SENDM instructions just discussed, except that the flits are marked as being tail flits, signaling the end of a message. Therefore, a complete SCMP message consists of the elements illustrated in Figure 2.5.

Understanding the types of network traffic in the SCMP architecture is critical to making high-level network design decisions. Simulation studies indicate that thread message latency is dominated by pointer arithmetic and not network latency. On the other hand, data message latency is dependent on both network traffic and the local node's memory access patterns, making system performance quite application dependent.

The SCMP network is designed to handle only dynamic traffic with no prior knowledge about traffic patterns, volume, or timing. As such, wormhole switching with virtual channel flow control has been selected as the architecture for the SCMP router. For its simple implementation and easier analysis, dimension-order routing will be used. A 2-D mesh topology has been chosen to keep wire lengths at an absolute minimum; however, implementing a folded torus requires only minimal changes to the routing circuitry and top-level wiring.

The SCMP tile, with multi-threaded processor and local memory, is connected to a router by the network interface unit (NIU). The NIU is responsible for packaging data from the processor and memory into network messages. As its name implies, the NIU provides the interface between raw data in the tile and the wormhole-switched router that makes up the network fabric. The router has great design flexibility, and this thesis focuses on the architectural and circuit-level impacts of the router design parameters. On the other hand, the NIU's structure is largely dictated by the router and processor/memory designs. A brief discussion of the NIU design and implementation will be given in section 4.2, but no further analysis of the NIU will be made.

Chapter 3
Related Research

While some constraints may change when considering on-chip networking, the broader context of parallel computer networks provides a wealth of existing research. This chapter consists of a review of the literature related to on-chip networking, beginning with the significant contributions from interconnection networks for parallel computers, specifically direct networks used with point-to-point connections.

3.1 Parallel Computer Networks

Built from commodity or custom microprocessors, parallel computer systems have leveraged interconnection networks to provide high-bandwidth message passing or tightly-coupled shared memory. Early message-passing computer systems included the Cosmic Cube [Seitz85], the Mosaic C system [Athas88], and the MIT J- and M-Machines [Noakes93, Fillo95]. These systems showed that fine-grained message passing was a capable platform on which to build a large, parallel computer system. In fact, the Mosaic network was so popular it was used as the interconnect for the Intel Paragon, Stanford DASH, and MIT Alewife machines [Boden95].

The Active Messages [vonEicken92] project demonstrated that the complexity of most message-passing implementations was creating unnecessary overhead in the network interface. If message headers contain information about the storage or processing of data, a significant reduction in the network complexity is achieved. The goal of active messaging is to decouple the processor-network interaction wherever possible, increasing the overlap of computation and communication.

The Pica architecture [Wills97] at Georgia Tech employed an active messaging style with integrated processor, network components and small amounts of local memory (4,096 36-bit words). Much of the SCMP architecture is an extension of the Pica system, except for the small memory footprint. In targeting future embedded DRAM fabrication capabilities, the SCMP architecture integrates large amounts of memory with processing and network circuitry.

In shared-memory systems, the interconnection network is responsible for transporting cache coherency data and control signals. Here, low latency and latency-tolerant architectures are paramount. Many small-scale multiprocessors have been built around a shared bus, limiting scalability. Larger shared-memory systems have been built using the non-uniform memory access (NUMA) paradigm with interconnection networks.

In the last decade, commodity interconnection networks have been developed for building parallel systems, marking a departure from the custom-designed systems of the past. Using the Parallel Virtual Machine (PVM) or Message-Passing Interface (MPI) programming paradigms, interconnects such as Myrinet [Boden95] and the Scalable Coherent Interface [James90] have enabled the construction of parallel systems from widely-available microprocessors.

3.2 Networks-On-Chip

As VLSI has advanced, researchers have begun to suggest the integration of multiple processor cores on a single chip. The RAW project at MIT is combining processing units with static and dynamic interconnects [Waingold97]. More than just a miniaturization from the parallel computer world, RAW is also integrating elements of reconfigurable computing to create a flexible computing solution. RAW uses both statically schedule and dynamically allocated network mechanisms [Taylor02]. The static network is treated as a programmable resource and message transmission is scheduled at compile time. The dynamic network is a conventional wormhole network. Physical interconnect channels in the RAW architecture carry multiplexed static and dynamic network traffic, with priority given to the scheduled, static network.

Dally and Towles [Dally01] proposed using a general-purpose interconnect to replace the ad-hoc global wiring found in modern VLSI designs. Leveraging plentiful on-chip resources, their sample design advocates using a folded-torus topology to reduce the average number of routing hops. Their design uses virtual-channel flow control with a very large flit width (300 bits) due to the lack of pin constraints found in multi-chip networks. The example network sketched by Dally and Towles is not application-specific; rather, it targets arbitrary client logic (microprocessors, DSPs, memory, etc.).

In [Guerrier00], Guerrier and Greiner propose an on-chip network architecture for the creation of Systems-On-Chip (SoC) designs. They argue that a switched network will lead to more scalable designs than the current bus-based designs in place today. Their prototype network uses packet-switching in a fat-tree topology, as opposed to the wormhole switching, virtual channel flow control in a 2-D mesh proposed in this thesis.

Guerrier and Greiner argue that wormhole switching alone results in under-utilized network resources in the presence of contention. This is true without virtual channels, as [Peh01] concludes as well. The crossbar switch used in [Guerrier00] is quite large (10x10), even though not all 100 switch connections are made. Nonetheless, it is likely that their design's cycle time will be limited by the large switch. The analysis in [Guerrier00] does include cost and performance tradeoffs; however, it does not provide power consumption data or detailed timing results. The area data provided is not parameterized and therefore would be difficult to extend beyond the design and 0.25μ implementation given. That said, the work in [Guerrier00] is significant in that it proposes a complete, generic architecture for building scalable SoC designs.

Jantsch, Tenhunen, and colleagues at the Royal Institute of Technology in Sweden have been developing a system design approach using Networks-On-Chip (NoCs) [Jantsch03, Kumar02, Soininen03]. Without committing to a specific interconnect topology or network design (switching, routing, or flow control), they propose connecting Intellectual Property (IP) blocks in a programmable fashion. Much of their work has focused on system- and platform-level design issues, and the work presented here could be easily applied to the design methodologies they have proposed.

3.3 Quantifying Performance and Implementation Costs

Researchers have developed a multitude of metrics to quantify network and system performance. Measuring latency and bandwidth in network components has been widespread since the first interconnection networks were proposed. An area of active interest has been the development of accurate delay models for routers of various classifications. Delay models seek to quantify the latency of a single network hop; that is, the time it takes one packet or flit to clear a router. Chien [Chien93] proposed a basic delay model for wormhole routers, which was later extended by Peh and Dally [Peh01] to account for pipelining and more practical virtual channel implementations. In section 5.1.1, the models of [Peh01] will be applied to compute the delay of several SCMP router design alternatives.

Chien [Chien93] also advocated the consideration of implementation complexity when designing interconnection networks. He used the gate count to quantify the complexity of several routing algorithms in a k-ary n-cube, wormhole-switched network. One drawback in using gate count to estimate implementation cost is the assumption that wires do not consume area or contribute to complexity. A few recent router designs have area information given, or it can be extracted by observing a photomicrograph of the implementation. Section 5.1.2 will extend the work of Chien to include wire and gate areas derived from VLSI models.

Peh and her colleagues at Princeton have begun developing power models for wormhole networks with virtual-channel flow control [Wang03]. Using circuit models for power consumption, they have obtained architectural-level estimates that appear to come within 10% of implementation results for several modern routers. With an ever-increasing emphasis on power efficient designs, it is critical to continue the development of power models for network architectures.

The power model in [Wang03] considers only dynamic (switching) power, which makes up the majority of the power dissipated in current fabrication technologies. The commonly used formula for dynamic power consumption is $P = 1/2\alpha C V_{dd}^2 f_{clk}$, where α is the switching activity, C is the switch capacitance, V_{dd} is the supply voltage, and f_{clk} is the clock frequency. In this thesis, as in [Wang03], power models will be determined by estimating α and C. Dynamic power modeling will be applied to the SCMP network design in section 5.1.3.

Chapter 4

Design and Implementation

The SCMP router has been designed for pipelined operation so that its clock cycle matches the speed of the SCMP processors. The network interface has been designed as two finite state machines controlling the inject and eject operations. Both the router and NIU were prototyped using the SystemC hardware design language to obtain a cycle-accurate simulation of the network subsystem. For detailed area estimates, the crossbar switch and flit buffers in the router have been designed in full custom VLSI layout. This chapter describes the detailed design and implementation efforts as part of this project.

4.1 Router

As Peh and Dally suggest [Peh01], modern router implementations are nearly all pipelined. The SCMP router consists of four substantial operations: decode+routing, virtual channel allocation, switch allocation, and crossbar traversal. These four operations make up a four-stage pipeline that serves as the basic router design. At the beginning of a flit's router traversal, buffers store the flit in case resource contention blocks its passing. The input buffering happens in parallel with the virtual channel and switch allocation stages, which are only used when a header flit enters the port. Figure 4.1 shows the router's composition and pipelined structure.

To keep the crossbar switch at a manageable size, the virtual channels are multiplexed across the five switch inputs. Having all the virtual channels connected directly to the switch results in less arbitration logic but a switch that could be prohibitively large. It has been shown that arbitration logic costs little in terms of power [Wang03] and, according to the delay models of Peh and Dally [Peh01], the switch arbitration stage is almost 50% faster than the virtual channel arbitration stage, leaving time for the simple arbitration between virtual channels.

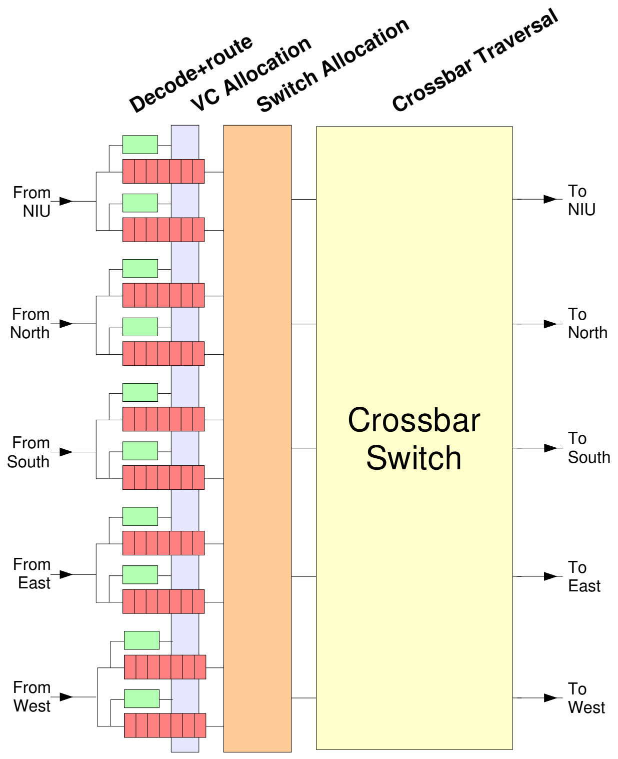


Figure 4.1: SCMP router with four pipeline stages. The input buffering occurs for all flits entering the port, while the decode+routing and virtual channel allocation stages are only used for header flits and operate in parallel with the buffering. Switch paths are allocated on a per-flit basis.

While a four-stage pipeline has been tentatively used for architectural analysis, the final, optimal pipeline design will come as a result of back-annotating timing information from layout or synthesis of the router components. Because the SCMP processor is not heavily pipelined (only five stages), it can be expected that the optimal router pipeline will not be very deep, and four stages is likely a good choice. In the following subsections, the major components of the router are discussed further and implementation details are given.

4.1.1 Decode+Routing

When the header flit of a message enters the router, its destination must be decoded and the correct route must be determined. When a header flit is detected, the X- and Y- offsets are decoded and the destination direction is obtained. The offsets each indicate the number of remaining hops in that direction, and the most-significant bit (MSB) of the offset indicates west or east in the case of the X-offset, and north or south in the case of the Y-offset.

Implementing dimension-order routing is straightforward. For a non-zero X-offset, the destination direction is either west or east depending on the MSB. If the X-offset is zero, the Y-offset is used in a similar manner when traveling north or south. A decrement or increment operation is used to update the offset for the next hop. When both X- and Y-offsets are zero, the destination is the local node's NIU. Other routing algorithms could easily be implemented in place of the dimension-order routing currently used.

Once a header flit has been decoded and routed, a virtual channel allocation request is placed in a small FIFO queue. The queue is necessary for the corner case when several small messages (i.e. only header, address, and tail flits) enter the same virtual channel. The queue depth is given by $\lceil B/3 \rceil$, where B is the number of flit slots in a virtual channel buffer. The current queue outputs are used in the second router stage to allocate virtual channels of the destination node.

4.1.2 Virtual Channel Allocation

Virtual channels of the destination router are allocated at the sending router. Because the SCMP network doesn't recognize network priorities, a simple round-robin arbiter is used. The allocation is done in a two-stage, separable manner, as illustrated in Figure 4.2. In the first stage, a single (valid) virtual channel is chosen from each input port. These virtual channel outputs are forwarded to the second stage, where each virtual channel of each output port chooses from the forwarded virtual channels. Separable allocation doesn't guarantee perfect matchings, but rather represents a reasonable compromise of performance and implementation complexity [Peh01].

Each of the allocation modules of Figure 4.2 uses the same generic structure to implement a round-robin arbiter. The circuit structure, shown in Figure 4.3

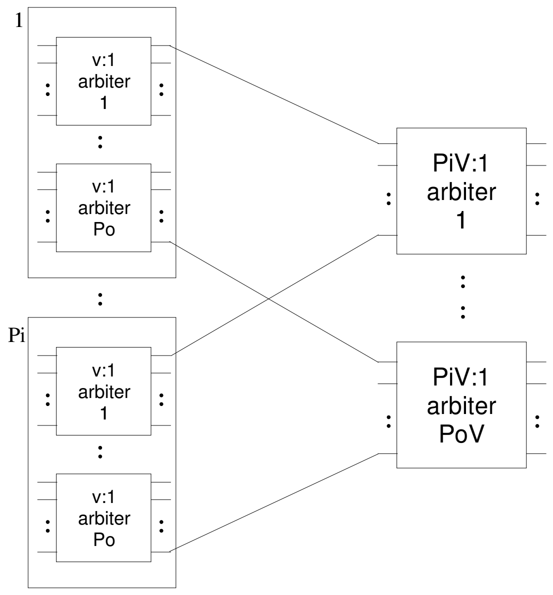


Figure 4.2: Separable virtual-channel allocator. In the first stage, each input port has a set of arbiters for each output port. The second stage arbiters are larger, having $p_i v$ inputs. These larger arbiters choose one of the chosen virtual channels from the first stage to receive control of the virtual channel on the receiving node.

is inherited from a simple matrix arbiter used when priorities are present in the network [Wang03]. In a matrix arbiter with m inputs, a binary matrix is constructed by setting the element of the i^{th} row and j^{th} column to 1 if the priority of input i is higher than that of input j. Because of the symmetric nature of the matrix, only $m(m-1)/2$ flip-flops are required to store priorities.

For a round-robin arbiter, m sets of priority matrices are kept, and the chosen matrix is rotated using a one-hot pointer. Because the matrix priorities are fixed, flip-flop storage elements are not required.

4.1.3 Switch Allocation and VC Multiplexing

In wormhole switching without virtual channels, the switch ports are allocated on a per-message basis. With virtual channels, however, switch ports are allocated for each flit. As with virtual channel allocation, a separable allocator is used to balance performance and implementation costs. In the first

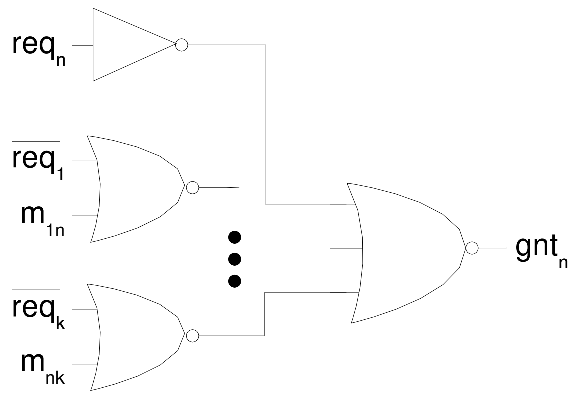


Figure 4.3: Matrix arbiter modified for use in round-robin operation. The matrix priorities $m_{i,j}$ are rotated depending on the last granted input.

stage, an allocated virtual channel from each input port is selected. The second stage selects an acceptable input port for each output port, should one exist. Again, the arbitration used in both stages is the round-robin arbiter of Figure 4.3. The entire switch allocation stage is diagrammed in Figure 4.4.

4.1.4 Crossbar Switch

The crossbar switch connects input paths to output paths, enabling the routing of flits through the network. A schematic figure of the SCMP crossbar switch is shown in Figure 4.5. Note that not all the connections are made in the crossbar. The missing connections aren't necessary due to the dimension-order routing. For instance, a message traveling from the south cannot then move in the east direction (flits are routed along the x-dimension first). With dimension-order routing, the total number of switch connections required in a 5-by-5 crossbar is 16.

Each switching element can be implemented in a variety of ways. The two most popular choices are transmission gates and tri-state buffers, which are both shown in circuit form in Figure 4.6. The transmission gate has been chosen in this thesis, due to its smaller size. The tri-state buffer should be used when faster switching speeds are required.

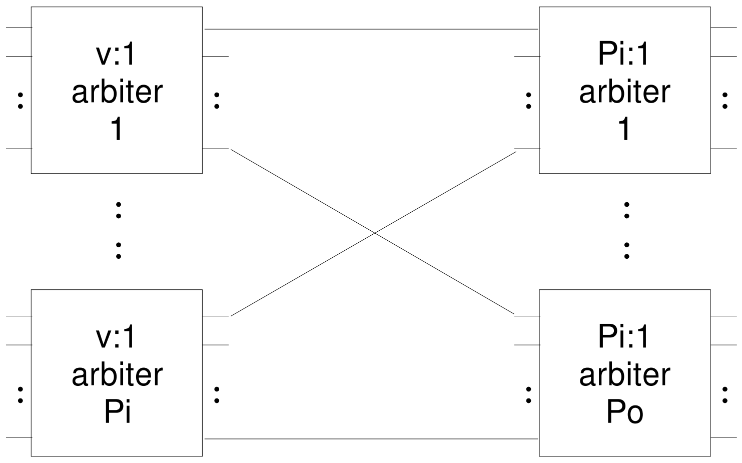


Figure 4.4: Separable switch allocation module. The arbitration circuits in the first stage choose a flit from each input port. The winning flit is forwarded to the appropriate output arbiter before going on to the switch.

The crossbar has been designed with the "Magic" VLSI layout package using the MOSIS deep-submicron scalable CMOS fabrication design rules (SCMOS_DEEP). Figure 4.7 shows the layout of a single transmission gate, while Figure 4.8 is the layout of a set of switches represented by a single connection point in Figure 4.5.

The wiring strategy for the crossbar switch layout is as follows. Metal layer one is used to bring input data signals vertically to the area of the switching elements. Metal layer two carries those input signals horizontally to the actual switches. The outputs are routed horizontally in metal layer three. This wiring strategy has been chosen to enable a compact design that can be reused as a standard switching cell used in forming the larger crossbar switch, as shown in Figure 4.9.

4.1.5 Virtual Channel Buffers

The virtual channel storage is implemented in first-in-first-out (FIFO) buffers termed "flit buffers". Each flit buffer consists of B buffer slots and pointers to mark the current buffer entry and exit slots. In modern router architectures, FIFO buffers are typically implemented using SRAM arrays [Wang03]. An

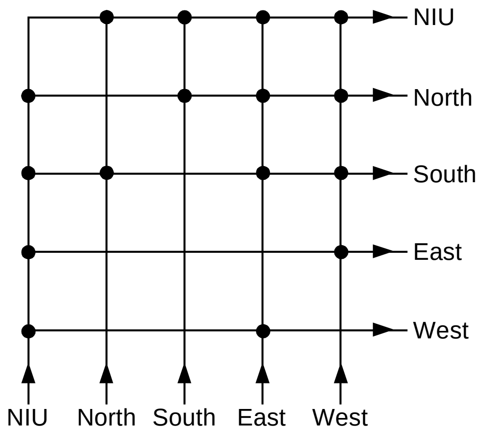


Figure 4.5: A 5-by-5 crossbar switch. Inputs run vertically and outputs run horizontally. Each (•) represents a set of connections made between inputs and outputs. Note that not all connections are required, due to dimension-order routing restrictions.

alternative approach, used in the MIT RAW project, implements the buffers using shift registers. The RAW project is an exception, however, and their choice is likely based on the lower storage requirements associated with the static network. As buffer sizes increase, the additional complexity of SRAM arrays is amortized.

Supporting purely dynamic traffic, the SCMP network will use SRAM arrays to implement virtual channel buffers. The SRAM cell used must have independent read and write ports, allowing simultaneous access to store a new

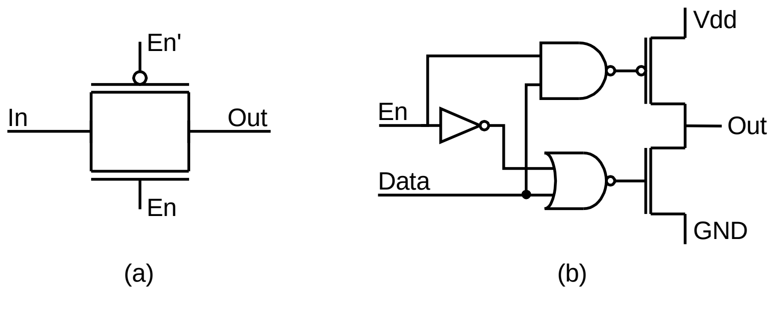


Figure 4.6: Circuit diagrams of (a) transmission gate and (b) tri-state buffer.

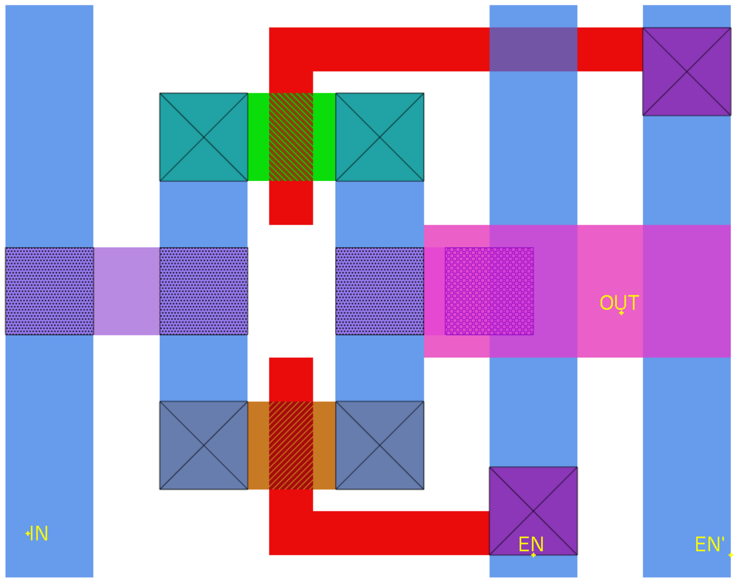


Figure 4.7: VLSI layout of a transmission gate.

flit and retrieve the oldest flit in the buffer. Unlike SRAM arrays used in cache blocks or other common applications, the virtual channel buffers don't need column or row decoders. That is, each row in the array corresponds to one buffer slot, and read and write pointers can be be implemented as one-hot words that are rotated to operate the FIFO buffer. Figure 4.10 shows a circuit diagram of both a "typical" single-port SRAM cell and the SRAM cell used here for the FIFO buffers [Weste94].

The VLSI layout of the FIFO SRAM cell is depicted in Figure 4.11. The outermost vertical wires (metal two) carry GND to the N-FETs in the inverters, while the top, horizontal wire (metal one) carries V_{dd} to the inverter P-FETs. The two inner vertical wires (metal two) carry the differential read pair, and the outer vertical wires carry the differential write pair. The read and write word lines run horizontally in polysilicon. This implementation is adapted from the notes found in [HorowitzNotes].

SRAM read and write operations use differential bit lines (bit_r/bitp_r and bit_w/bitp_w in Figures 4.10 and 4.11). Part of the overhead associated with SRAM arrays accommodates precharging the bit lines and subsequently sensing and amplifying the small changes in the differential signals. The SRAM

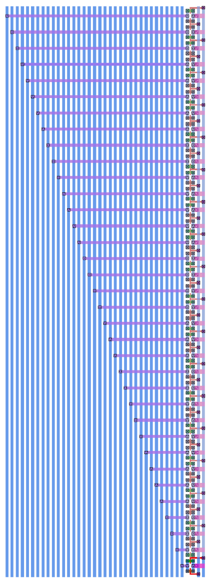


Figure 4.8: Layout of a set of transmission gate switches.

buffer is organized as a B x F array, where B is the number of buffer slots and F is the flit size. Each of the F columns requires a preamplifier and sense amp, as shown in the schematic of Figure 4.12. The layout of a complete 6-by-34 flit buffer is shown in Figure 4.13.

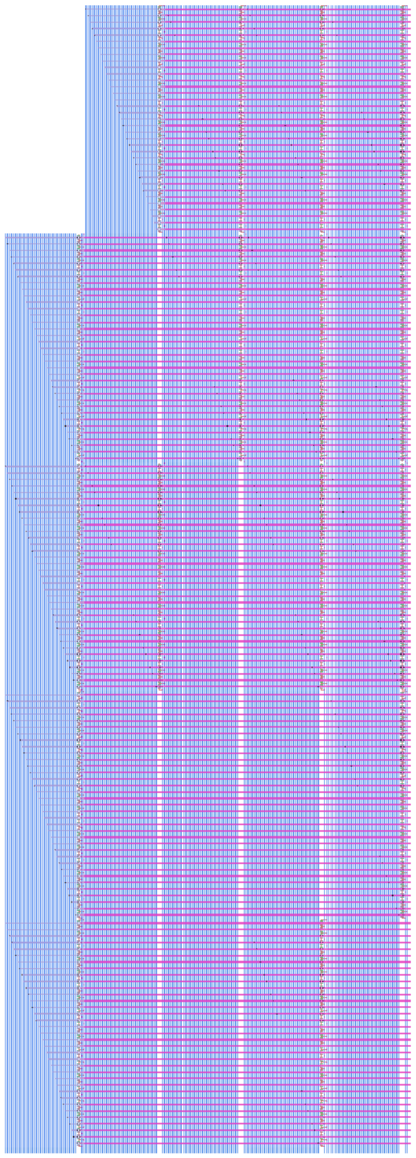


Figure 4.9: VLSI layout of a 5-by-5 crossbar switch.

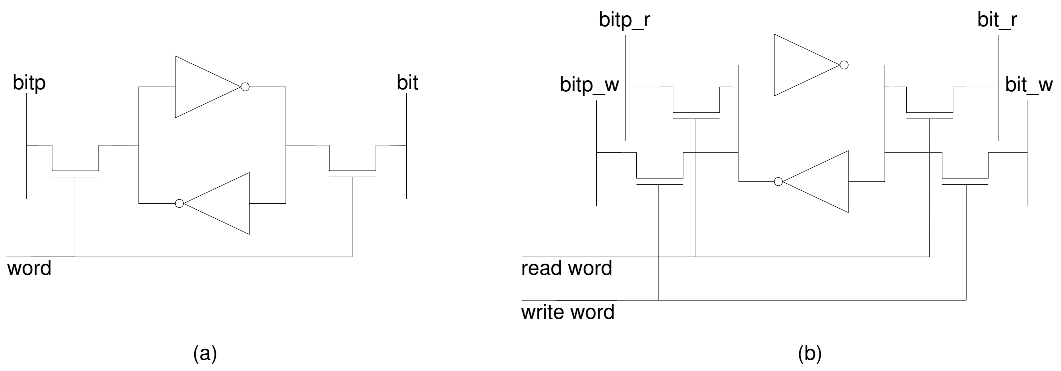


Figure 4.10: Circuits for (a) conventional 6-T SRAM cell and (b) fully dual-ported SRAM cell with independent read and write.

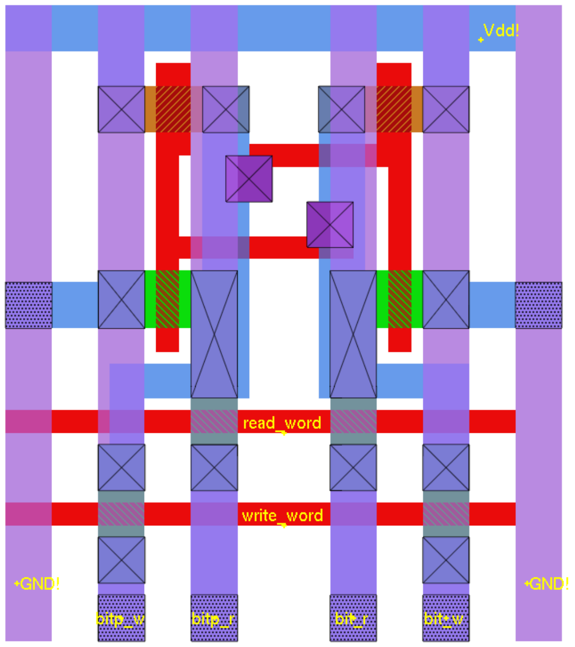


Figure 4.11: VLSI layout of fully dual-ported SRAM cell.

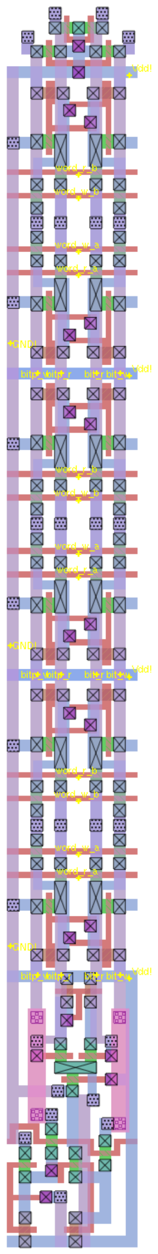


Figure 4.12: VLSI layout of six SRAM cells and associated precharge and sense amplifier circuitry.

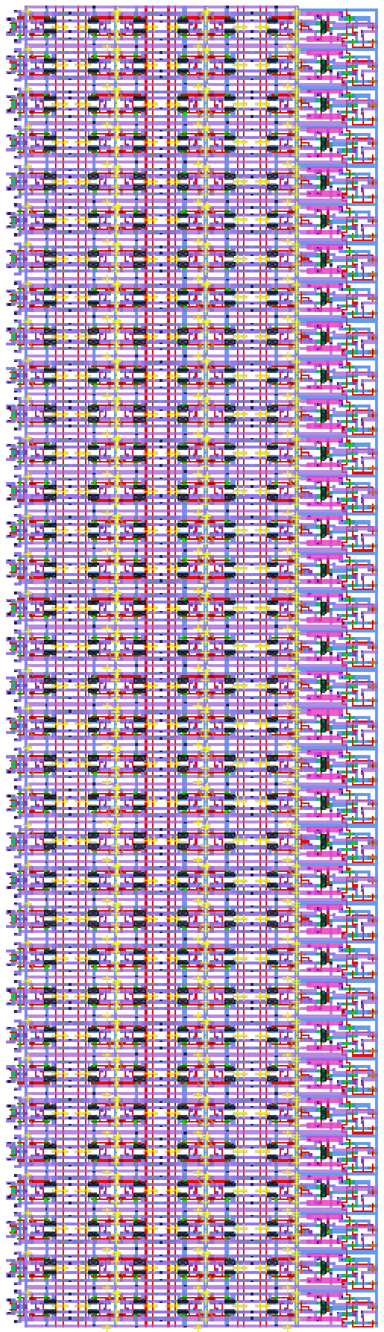


Figure 4.13: VLSI layout of 6 row by 34 column SRAM array.

4.2 Network Interface Unit

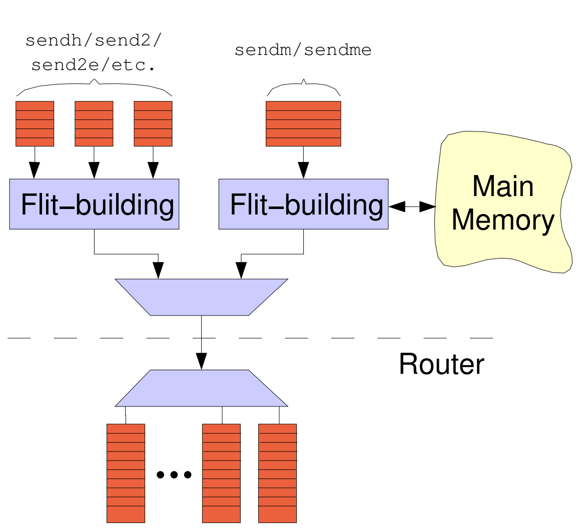
The NIU serves as glue logic between the router and the rest of the SCMP tile components. The two essential operations of the NIU are the ejection of flits from the network and injection of flits into the network. The two mechanisms are completely decoupled and are implemented in separate modules. Figure 4.14 contains a diagram of the inject and eject operation in the NIU.

The injection side receives control signals from the pipeline, indicating when messages are to be started or a new flit is to be transmitted as part of a message. Within the inject operation, two components can be identified. The first component receives message creation commands (`sendh`) and single or double flit-sending commands (`send`, `sende`, `send2`, and `send2e`). With these commands come the values fetched from the context registers by the pipeline, and both opcode and register values are buffered into small FIFOs.

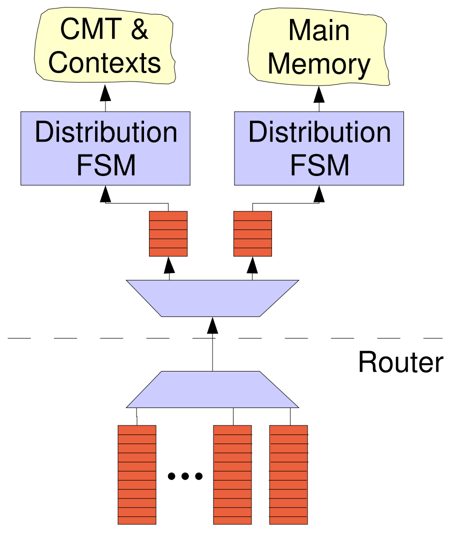
The second component of the injection side receives commands for sending blocks of memory. Because the memory blocks likely exceed a single word, the entire operation will be completed over a number of clock cycles. The base memory address is received in the initial command, along with a length and address stride. The NIU keeps an active memory address pointer that is updated as each memory word is fetched.

The ejection side is similarly composed of two finite state machines corresponding to the two message types: thread and data. Thread messages are destined for an FSM that will allocate a new thread and send register values to the context blocks. Data messages are sent to a simple FSM, which will transfer data values to memory. The FSMs are both necessary to provide control flow in the event access to contexts or memory is temporarily not allowed.

The NIU is not a component that affects performance greatly. As glue logic, few design alternatives exist for implementing the NIU. For these reasons, the NIU will not be discussed further when analyzing the performance and implementation of the network.



(a) Inject operation



(b) Eject operation

Figure 4.14: Network Interface Unit (NIU) structure and operation.

Chapter 5

Analysis

Numerous design alternatives exist for the SCMP router, as presented previously. In this chapter, the number of virtual channels and the depth of the flit buffers will be varied, and the resulting performance and cost tradeoffs will be analyzed. A variety of analysis tools have been developed and are put to use here. First, analytical models will be applied to derive delay, power, and area estimates. Second, simulation tools will be used to obtain both pure network performance information and application-based results.

The methods applied in this chapter form the framework for analyzing on-chip interconnection networks. The results presented here are applicable to the SCMP computer; however, the methodology can be applied to many future on-chip networks, especially permutations of the SCMP design.

5.1 Analytical Models

Analytical models provide tools for understanding the network's behavior beyond what may be exhibited in a set of benchmarks or other simulation-based tests. However, various assumptions and idealizations must be made to construct an analytical model, which must be accounted for when inferring behaviors about the actual network. In this thesis, three analytical models will be constructed: delay, power, and area.

5.1.1 Delay

The delay model used in this thesis comes from the work of Peh and Dally [Peh01]. The purpose of an analytical delay model is to predict timing behaviors of the various router pipeline stages from an architectural level. That is, without going through a full implementation and back-annotation, a delay model can provide reasonable estimates of component timing requirements. With this information, an optimal pipeline structure can be formulated early in the design process.

In the model of [Peh01], p is the number of ports on the crossbar switch, F is the flit width, and v is the number of virtual channels per physical channel. Peh and Dally point out that a realistic model should multiplex several virtual channels across a single crossbar port. Otherwise, the crossbar grows to a prohibitive size, and switch traversal cannot take place in one clock cycle.

Router operation consists of four stages: decode+routing, vc allocation, switch allocation, and crossbar traversal. These four stages are atomic and cannot be easily pipelined. Therefore, the simplest pipeline architecture uses the four stages shown in Figure 4.1. Each atomic module i has a latency of t_i and overhead h_i in the notation of [Peh01]. Latency corresponds to the time to process inputs in the module, while overhead is the time required to store module state information before the next input set can begin processing.

Peh and Dally's delay model uses the method of logical effort [Sutherland91]. A module's delay consists of two components: effort delay (T_{eff}) and parasitic delay (T_{par}). Effort delay is defined as the sum of logical effort and electrical effort. Logical effort is the ratio of a module's delay to the delay of an inverter with the same input capacitance. Electrical effort is the module's fanout, or ratio of output to input capacitance. Parasitic delay is the delay due to the internal capacitance of a circuit relative to the delay caused by the parasitic capacitance of a similar-sized inverter. The delay model of Peh and Dally uses a technology-independent timing parameter of τ, the delay of an inverter with identical input capacitance.

In [Peh01], the example of a single inverter driving four inverters is given. The logical effort of an inverter is 1, while the electrical effort is 4 (fanout). Therefore, the effort delay is 1 x $4 = 4$ and the parasitic delay is 1. The total circuit delay, relative to the parameter τ is then $4 + 1 = 5$. Peh and Dally define this circuit's delay as $\tau_4 = 5\tau$, and claim that a nominal clock cycle to use is $20\tau_4$.

Table 5.1 summarizes the circuit delay for the four atomic modules of the router model. For details on the derivation of these equations, consult [Peh01]. The delays are expressed as functions of v, the number of virtual channels. Here, the number of ports p is fixed at 5 and the flit width F is 34 bits. As in [Peh01], the delay for routing and decoding is assumed to be one cycle, or 100τ.

Table 5.1: Router modules' delay as function of the number of virtual channels v and delay parameter τ.

Module	Delay equations (τ)
Route+decode	Assumed to be 100
Virtual-channel allocator	$t_{VC} = \frac{33}{2}\log_4 p + 33\log_4 v + \frac{125}{6}$ $h_{VC} = 9$
Switch allocator	$t_{SL} = \frac{23}{2}\log_4 p + 23\log_4 v + \frac{125}{6}$ $h_{SL} = 9$
Crossbar traversal	$t_{XB} = 9\log_8\left(F\left\lfloor\frac{p}{2}\right\rfloor\right) + 6\lceil\log_2 p\rceil + 6$ $h_{XB} = 0$

Figure 5.1 shows the pipeline timing of several router configurations with varying numbers of virtual channels (VCs) per physical channel. The VCs are evenly divided among thread and data messages. One cycle is assumed to be $20\tau_4 = 100\tau$. Observe that configurations having less than ten total VCs fit into four cycles, while ten virtual channels require an additional cycle for VC allocation.

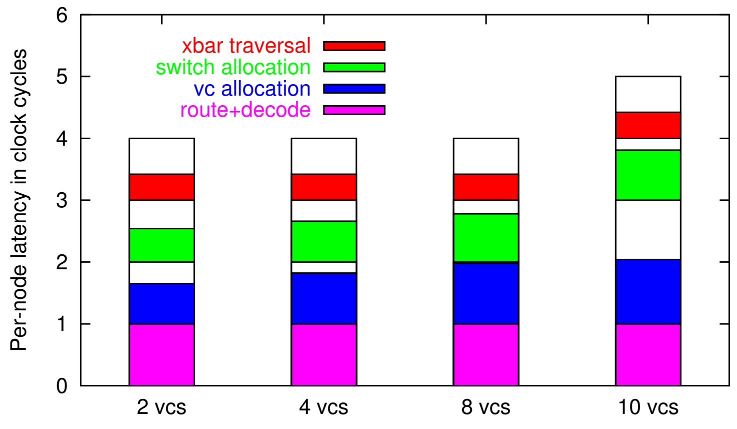


Figure 5.1: Router pipeline timing estimates for various numbers of virtual channels.

5.1.2 Area

While the router is not expected to consume as much area as other parts of the SCMP system, its size is not negligible. The major consumers of area in the router are the flit buffers and the crossbar switch. Simply counting transistors does not give a sufficient estimate of area, however, as the crossbar is primarily consumed by wire area. Because the flit buffers are fully dual-ported (see Section 4.1.5), conventional single-ported SRAM area models are insufficient. Rather, the area of flit buffers and crossbar must be determined empirically.

The area of the router components is parameterized by using VLSI cells that can be repeated to build larger modules. The VLSI design uses the MOSIS deep-submicron scalable CMOS (SCMOS_DEEP) design rules, which parameterize feature sizes by λ. For a given fabrication process, there is a one-to-one mapping between λ and the fabrication feature size, as in 0.5μ or 0.18μ.

For the flit buffers, the basic building block contains two SRAM cells, each of the form in Figure 4.11. The cells are built into a column, as in Figure 4.12, where precharge and sense amplifier circuits are added. The columns are then aligned together to build the complete SRAM buffer, as in Figure 4.13.

The two SRAM cells occupy 44λ wide by 102λ tall. The precharge and sense amplifier circuitry add 20λ and 94λ in height, respectively. Therefore, the total area required for an SRAM buffer of B F-bit words is

$$p\left(44F \text{ x } \left(\frac{B}{2}102 + 114\right)\right) \; \left[\lambda^2\right].$$

For the SCMP network, F is fixed at 34 bits. Figure 5.2 shows flit buffer area for a number of virtual channel configurations. For even comparison, the total amount of buffer storage is held fixed per port. That is, for two virtual channels per port, buffers holding eight, sixteen, or thirty-two flits are used. For four virtual channels, buffers with four, eight, and sixteen flits are used, etc. The area numbers given in Figure 5.2 are the total storage required for all five ports.

For the crossbar switch, the basic building block is two of the transmission gates of Figure 4.7. The area of this set of gates is 26λ wide by 46λ tall. Using

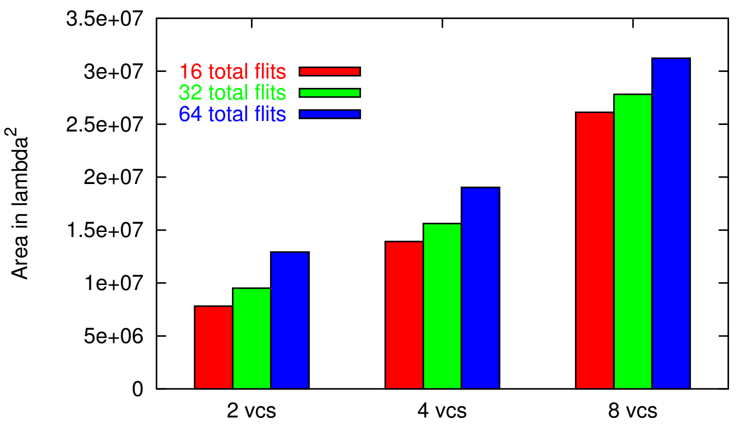


Figure 5.2: The area of SRAM flit buffers in various configurations.

the strategy described in Section 4.1.4, a complete p-by-p crossbar for $W = F + 1$-bit flits requires $p(26 + 7W)\lambda$ by $p(22W + 4)\lambda$.

In the case of the SCMP router, W must be one larger than the usual 34 bits to carry a valid signal. Other router definitions often include the valid signal as part of the basic flit format. For $p = 5$ and $W = 35$, the total area required is $1355\lambda\ x\ 3870\lambda = 5.24 \times 10^6\lambda^2$. Note that this is smaller than the area required for even two virtual channels with only eight flits per buffer, and is nearly six times smaller than a router with eight virtual channels and eight flits per buffer.

Area estimates have not been generated for the routing and arbitration logic, as it is expected that they will not require much area in comparison with the flit buffers or even crossbar.

5.1.3 Power

As power consumption has become an increasingly important factor in microprocessor design, so has the need to model and optimize power performance at the architectural level. Several tools [Brooks00, Duarte02] have recently been developed to estimate power consumption in superscalar, out-of-order processors without requiring a detailed gate-level simulation. With these tools, architects are able to explore more of the design space with regards to power.

Until recently, such tools were not available for interconnection networks. In [Wang03], Wang, Peh, and Malik describe a power estimation method for routers such as those used in the SCMP system. They model dynamic power with the commonly used formula $P = 1/2\alpha C V_{dd}^2 f_{clk}$, where α is the switching activity, C is the switching capacitance, V_{dd} is the supply voltage, and f_{clk} is the clock frequency. Where V_{dd} and f_{clk} will generally be set by process technology or other architectural features, C and α must be determined as part of the router architecture and operation.

In [Wang03], the authors showed that the primary power consumers were flit buffers and the crossbar switch. For router architectures requiring large arbitration circuits (e.g. 19:1 arbitration in the Alpha 21364 router [Mukherjee02]), arbitration power cannot be ignored. With dimension-order routing, the SCMP arbitration circuitry is significantly smaller (4:1) and the corresponding arbitration power is negligible. In this section, the power consumption of the flit buffers and crossbar switch will be modeled in the context of the SCMP router.

The switch activity α will be treated as an independent variable in the power consumption of the router. The flit arrival rate, P_f and the probability of a signal changing value, P_{fac}, combine to form α. Thus, the task of estimating power consumption is to model the capacitance.

Capacitance estimation is a well studied topic in the context of CMOS transistors. Using the theory presented in [Weste94], the capacitance of various circuit components will be estimated. Three elements contribute to the total capacitance of a CMOS circuit: gate, diffusion, and wire capacitance. The gate capacitance is formed by the gate conductor and the holes or electrons in the substrate. For the purposes of architectural power modeling, the gate capacitance is estimated as

$$C_g = C_{ox} \times W \times L + W \times C_{gso} + W \times C_{gdo} + (2L \times C_{gbo}), \tag{5.1}$$

where W and L are the width and length of the gate channel, respectively, C_{gso}, C_{gdo}, and C_{gbo} are the capacitances formed from fringing fields and overlapping materials. C_{ox} is the capacitance of the thin-oxide material, given by

$$C_{ox} = \frac{\epsilon_0 \epsilon_{SiO_2}}{t_{ox}}. \tag{5.2}$$

The value of ϵ_{SiO_2} is 3.9 and ϵ_0 is the permittivity of free space, 8.854×10^{-14}. t_{ox} is the thickness of the thin-oxide, obtained from SPICE model parameters.

The second capacitance value used is the diffusion capacitance, which is formed by the voltage between the diffusion region and substrate. Based on the theory developed in [Weste94], the diffusion capacitance is

$$C_d = C_j \times ab + C_{jsw} \times (2a + 2b), \tag{5.3}$$

where a and b are the width and height of the diffusion region, respectively. C_j is the junction area capacitance, and C_{jsw} is the capacitance of the side walls of the diffusion region, both obtained from SPICE models. Table 5.2 lists the capacitance values taken from the TSMC 0.18μ process available from MOSIS [TSMC18].

Table 5.2: Capacitance parameters from TSMC 0.18μ process [TSMC18]

Parameter	Value	Units
t_{ox}	$4.1\text{E}-9$	m
C_j	$9.775464\text{E}-4$	F/m^2
C_{jsw}	$2.244709\text{E}-10$	F/m
C_{gdo}	$7.32\text{E}-10$	F/m
C_{gso}	$7.32\text{E}-10$	F/m
C_{gbo}	$1\text{E}-12$	F/m

Finally, the capacitance of wires used to route components of the chip must be determined. While complex wire capacitance models have been developed for many years [Weste94], a simple approximation will be used for quick estimates. The wire capacitance is the sum of a parallel plate and fringing field components, modeled as

$$C_w = C_{pp} + C_{fringe} = \frac{w\epsilon_{di}}{t_{di}} + \frac{2\pi\epsilon_{di}}{\log(t_{di}/H)}. \tag{5.4}$$

For wires of minimum width, as is the most common case, a rough approximation to the above expression is poly wires (2λ width) are $0.25fF/\mu$, metal 1 wires (3λ width) are $0.3fF/\mu$, and metal 2 wires (4λ width) are $0.25fF/\mu$ [Ho01]. These values will be used, denoted C_{wp}, C_{wm1}, and C_{wm2}, respectively.

As discussed in section 4.1.5, the flit buffers in the SCMP router are implemented as FIFO SRAM buffers with independent read and write ports. Based on the models of [Zyuban98] and [Wang03], the power consumption of a FIFO SRAM buffer is

$$\begin{aligned} P_{buffer} &= f_{clk}\left(5P_f\left(E_{write}+E_{read}\right)\right) &(5.5)\\ E_{write} &= E_{wl}+FP_{fac}\left(E_{bw}+E_{cell}\right) &(5.6)\\ E_{read} &= E_{wl}+FP_{fac}\left(E_{br}+2E_{chg}+E_{amp}\right), &(5.7)\end{aligned}$$

where $E_x = C_xV_{dd}^2$, and

$$\begin{aligned} L_{wl} &= F\left(w_{cell}+2\left(P_r+P_w\right)d_w\right) &(5.8)\\ L_{bl} &= B\left(h_{cell}+\left(P_r+P_w\right)d_w\right) &(5.9)\\ C_{wl} &= 2FC_g+C_g+C_d+C_{wp}L_{wl} &(5.10)\\ C_{bw} &= BC_d+C_d+C_{wm1}L_{bl} &(5.11)\\ C_{br} &= BC_d+C_g+C_a+C_{wm1}L_{bl} &(5.12)\\ C_{cell} &= 2\left(P_r+P_w\right)C_d+2C_d+2C_g &(5.13)\\ C_{chg} &= C_g &(5.14)\\ E_{amp} &= \frac{1}{8}V_{dd}/f_{clk}\left(0.0005\right). &(5.15)\end{aligned}$$

As before, F is the flit width and B is the size of the buffer in flits. P_r and P_w are the number of read and write ports, respectively, which are both 1 in this case. w_{cell} and h_{cell} are the width and height of the SRAM cell, as given in the previous section (5.1.2). d_w is the wire spacing between the read and write word and bit lines.

This model has numerous shortcomings, most notably that the transistors are all assumed to be of the same, minimum size. In the case of the word line driver transistors, this is most likely in error. Therefore, the power estimates for the flit buffers are most likely undersized by a nominal amount. Further work is required to integrate the driver size into the model and observe the change in estimated power.

The power consumption of the crossbar switch is modeled in a similar fashion. Maximum power is consumed when all five switch outputs receive flits. When fewer ports are used, the power scales linearly, leaving

$$P_{crossbar} = 5 P_f P_{fac} \times F \left(E_{xb\_in} + E_{xb\_out} \right), \tag{5.16}$$

where

$$\begin{aligned} L_{in} &= 5 W w_t & (5.17) \\ L_{out} &= 5 W h_t & (5.18) \\ E_{xb\_in} &= C_{xb\_in} V_{dd}^2 = \left(5 C_{in\_cnt} + C_g + C_d + C_{wm1} L_{in} \right) V_{dd}^2 & (5.19) \\ E_{xb\_out} &= C_{xb\_out} V_{dd}^2 = \left(5 C_{out\_cnt} + C_g + C_d + C_{wm2} L_{out} \right) V_{dd}^2. & (5.20) \end{aligned}$$

Here, W is the port width (F+1 in SCMP), and w_t and h_t are the width and height of the input/output routing lines. C_{in_cnt} and C_{out_cnt} are the capacitance of the input and output connectors, respectively. The latter two capacitances are estimated as the coupling capacitance of metal 1 to metal 2, using information found in the MOSIS test results [TSMC18].

The total router power is then pieced together in

$$P_{router} = 5 P_{buffer} + P_{crossbar}. \tag{5.21}$$

It is worth explaining that the number of virtual channels do not directly play a role in the power consumed in the router. Because only one virtual channel is read or written per port per cycle, the consumed power is determined by the number of ports, and not the number of virtual channels. Indirectly, however, the virtual channels do factor in the power dissipation, as the depth of the flit buffers, B, is reduced as the number of virtual channels increases.

Figure 5.3 illustrates the power consumed in the SCMP router for various flit buffer sizes. To create quantitative results, the TSMC 0.18μ process is used with a frequency of $1.2GHz$ and V_{dd} of $1.8V$. λ is 0.10μ.

Several interesting observations can be made from the plots in Figure 5.3. First, the power consumption per router is quite small (less than 1 W). In comparing these results with the Alpha 21364 power consumption in [Wang03], the SCMP router consumes nearly an order of magnitude less power (6W for the 21364). The 21364 contains a much larger router, however, having 8 input and 7 output ports, much larger flit buffers, and buffers with two read ports instead of one. While each SCMP router may not consume much power, consider that sixty-four SCMP routers would consume 60W, not including power in the signal lines connecting the routers.

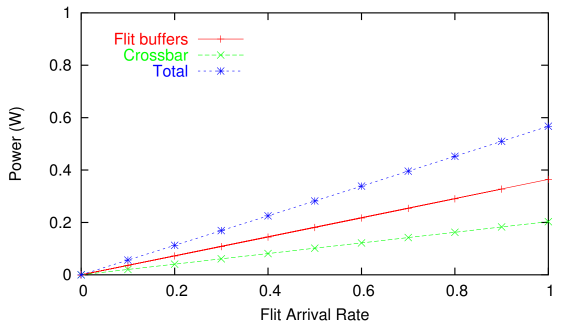
A second observation is that, as in [Wang03], flit buffer power consumption exceeds the crossbar power by a 3:1 ratio. The flit buffers therefore account for the majority of area and power, emphasizing their cost to the router design. In the next section, the performance tradeoffs of various virtual channel arrangements are considered.

5.2 Simulation Models

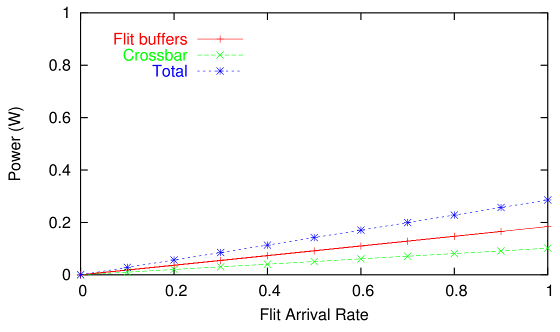
Where the previous two sections have given implementation "costs" in the form of area and power, the design analysis would be incomplete without considering the performance impacts of various configurations. As mentioned in the introduction, two general categories of performance evaluation will be used. The first simulates a synthetic network load to stress the router components alone, while the second uses application benchmarks to give an indication of the impact router design has on software performance.

5.2.1 Synthetic Tests

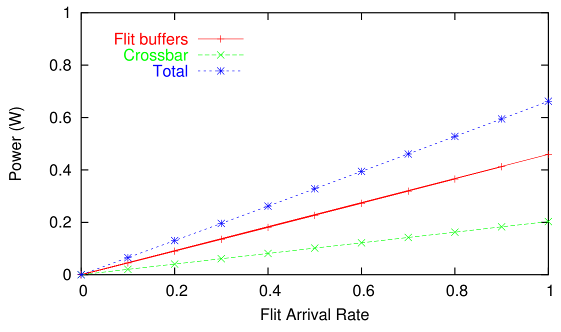
The synthetic tests consisted of sending 16,000 messages into the network, and measuring the message latencies upon reception. The messages were a fixed length of twenty flits (as in [Dally92]), and source and destination nodes were uniformly distributed. Tested network dimensions were: 2x2, 4x4, and 8x8. The results from each size will be treated independently, as the network behaviors are quite different as the number of nodes increases.



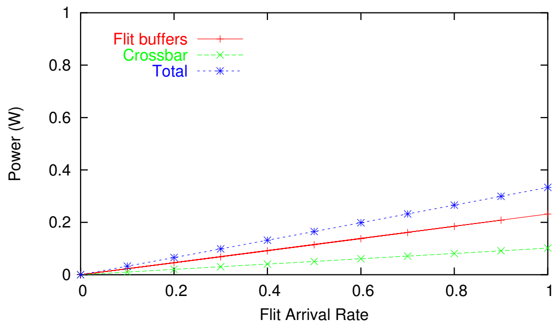
(a) Maximum power consumption with 4 total flits per port.



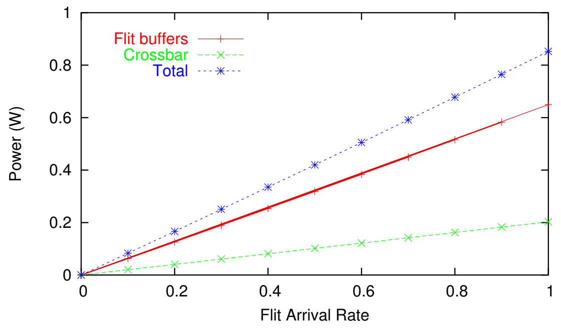
(b) Average power consumption with 4 total flits per port.



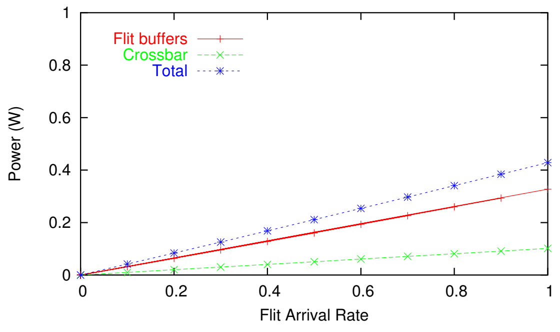
(c) Maximum power consumption with 8 total flits per port.



(d) Average power consumption with 8 total flits per port.



(e) Maximum power consumption with 16 total flits per port.



(f) Average power consumption with 16 total flits per port.

Figure 5.3: Power consumption with various router configurations.

For each size network, two sets of comparisons will be made. First, the number of virtual channels per port will be analyzed to observe the impact VCs have on latency and throughput. Second, the size of the flit buffers will be varied to observe their effects. In both cases, the Burton Normal Form (BNF) plot will be used to illustrate the behaviors. BNF plots average message latency versus a normalized accepted traffic. Accepted traffic is defined as the number of flits received per unit time and node. The normalization is done relative to $2B/(NF)$, where B is the bisection bandwidth, N is the number of nodes, and F is the flit width. For the 2-D mesh, B is $2\sqrt{N}F$.

In Figures 5.4 through 5.9, the synthetic network results are plotted in BNF form. Note that in varying the number of virtual channels, the depth of the flit buffers is adjusted to keep the total flit storage space constant. This becomes a factor in, for instance, Figure 5.4(c), where only 2 flits per vc buffer are insufficient storage. In general, for lower network sizes, the depth of the flit buffers appears more significant than the number of virtual channels.

As the network size grows, the number of virtual channels becomes more important. This observation makes practical sense, as messages pass through more intermediate nodes to reach their destination. Those intermediate nodes must pass more traffic than in the smaller network sizes, and the presence of more virtual channels will mitigate the impact of blocked messages.

5.2.2 Application Benchmarks

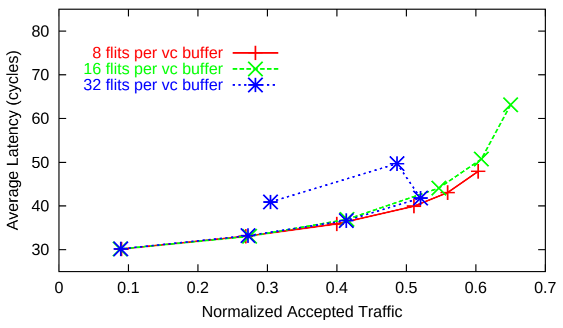
While synthetic network tests may illustrate router operation exclusively, application benchmarks must be considered when analyzing network performance. The assumptions made in conducting synthetic tests (fixed message size, uniform traffic) are in contrast with most applications, and therefore the results from application code are of considerable importance.

At this time, four applications are available for the SCMP system: FFT, IFFT, Median Filter, and Matrix Multiply. For the FFT and IFFT, a 2-D filter is performed on a 256x256 image. The median filter is also applied to a 256x256 image, and the matrix multiply kernel operates on 256x256 matrices. As with the synthetic tests, separate results are presented for networks of 2x2, 4x4, and 8x8 nodes. For each network size, the thread and data latencies are considered separately. Separate analysis is performed because the message sizes differ greatly between the two types, and therefore the latencies also differ.

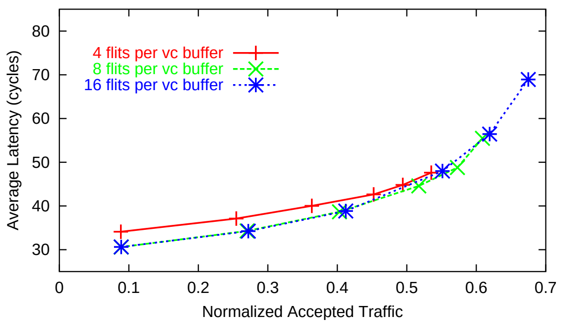
The simulation results are shown in Figure 5.10 through Figure 5.15. One immediate observation is that message latencies are roughly invariant to the size and number of VCs for small network sizes. This is most likely a function of the style of programming, where a data parallel paradigm is used. For smaller network sizes, each node handles more data by itself, and hence network activity is reduced. As system size increases, the network traffic increases and the impact of VC configuration is more visible.

In 2x2 and 4x4 systems, the number and size of virtual channels do not impact performance *in these benchmarks*. However, given the limited number of application codes available today, no absolute conclusions should be drawn from these results. In 8x8 configuration, the number of virtual channels do impact the performance of message latency in these tests, especially the median filter and matrix multiply.

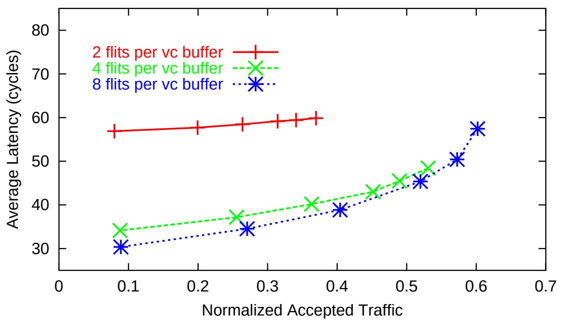
An additional note should be added about latency in the context of full system simulation. In sending thread messages, the latency includes time spent in pointer arithmetic, memory fetches, etc. as the message is built. More analysis of the applications and their composition is necessary to determine the impact these software-created latencies have on overall latency.



(a) 2 total virtual channels

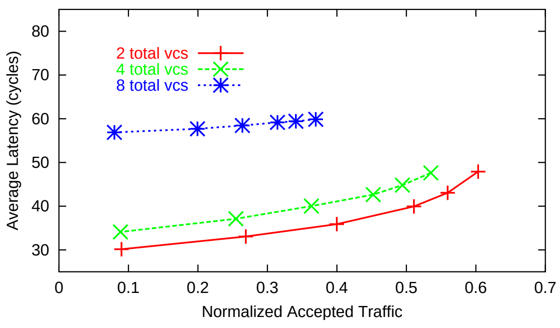


(b) 4 total virtual channels

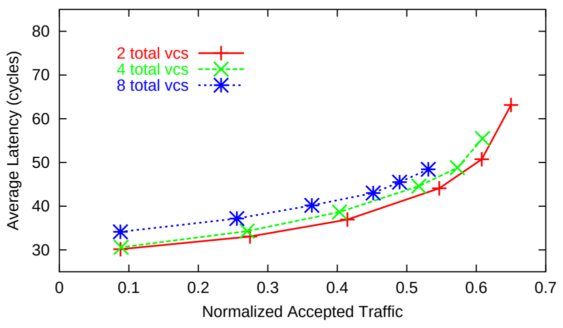


(c) 8 total virtual channels

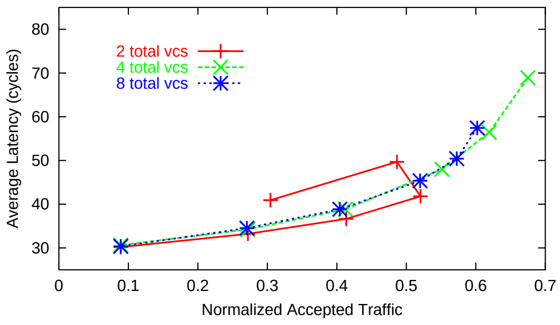
Figure 5.4: Latency vs. accepted traffic for 2x2 network with different numbers of virtual channels per port.



(a) 16 buffer slots per port

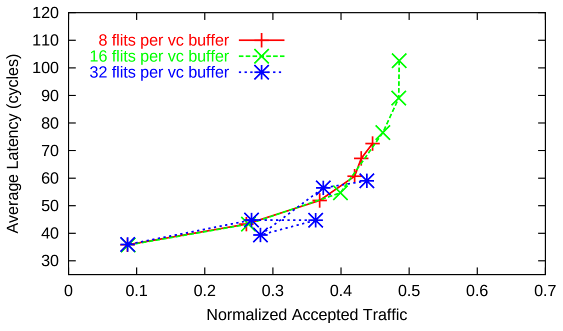


(b) 32 buffer slots per port

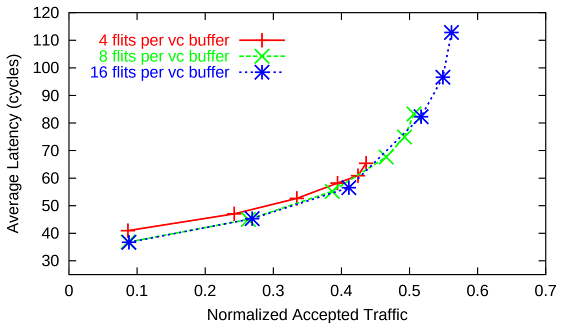


(c) 64 buffer slots per port

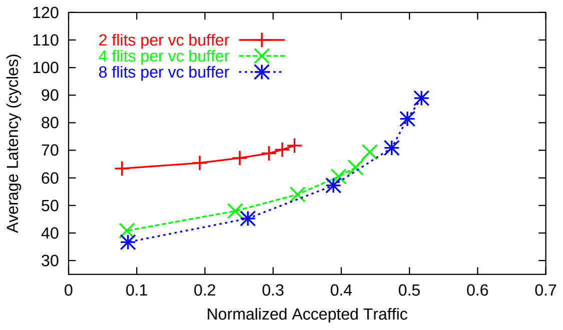
Figure 5.5: Latency vs. accepted traffic for 2x2 network with various amounts of buffer space.



(a) 2 total virtual channels

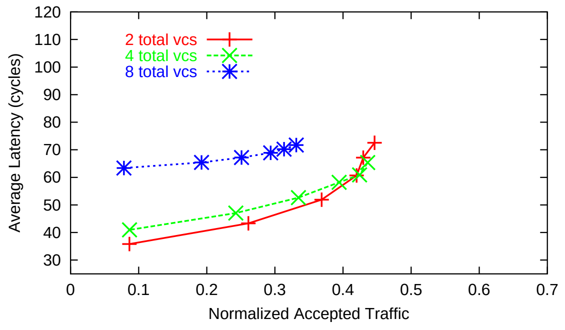


(b) 4 total virtual channels

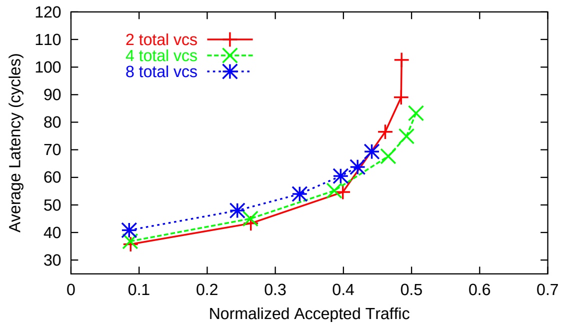


(c) 8 total virtual channels

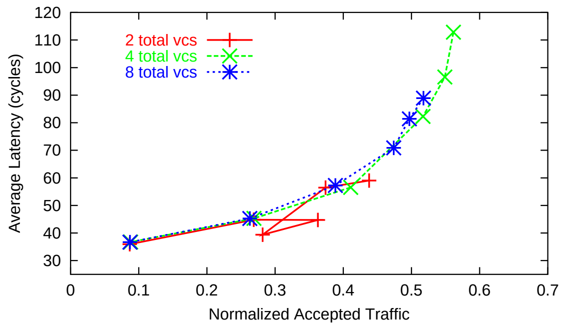
Figure 5.6: Latency vs. accepted traffic for 4x4 network with different numbers of virtual channels per port.



(a) 16 buffer slots per port

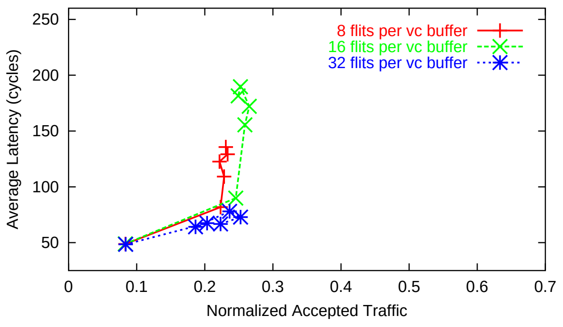


(b) 32 buffer slots per port

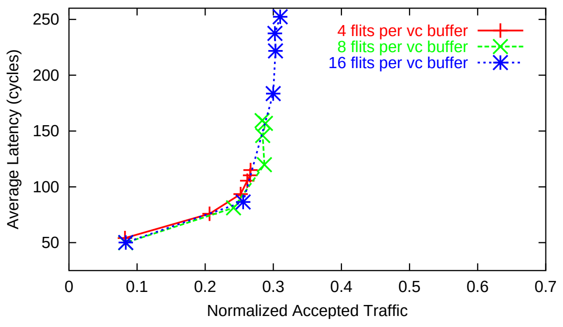


(c) 64 buffer slots per port

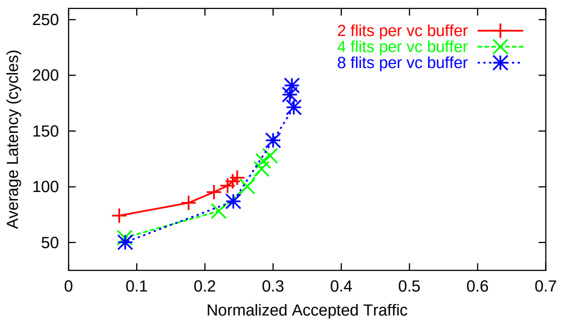
Figure 5.7: Latency vs. accepted traffic for 4x4 network with various amounts of buffer space.



(a) 2 total virtual channels

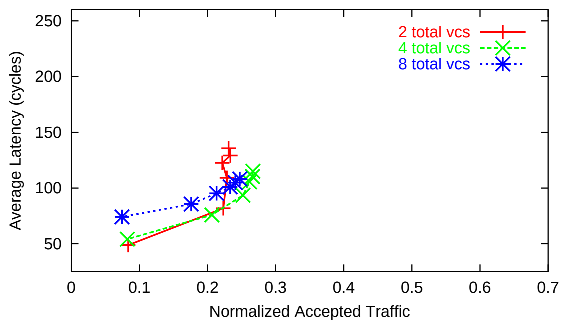


(b) 4 total virtual channels

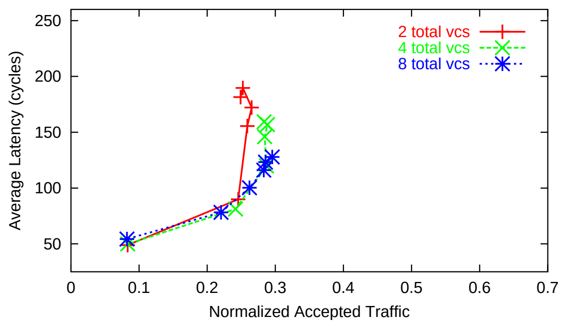


(c) 8 total virtual channels

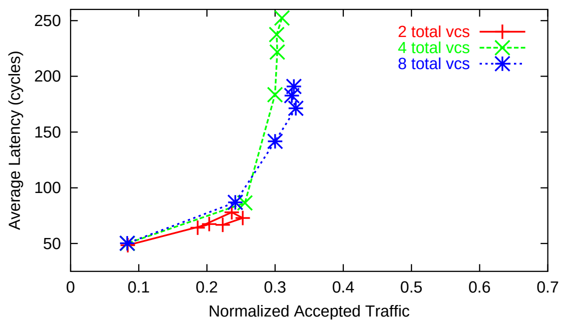
Figure 5.8: Latency vs. accepted traffic for 8x8 network with different numbers of virtual channels per port.



(a) 16 buffer slots per port

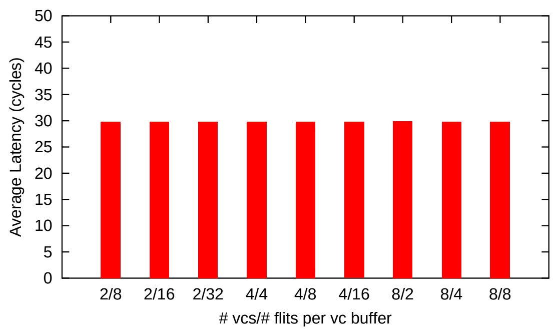


(b) 32 buffer slots per port

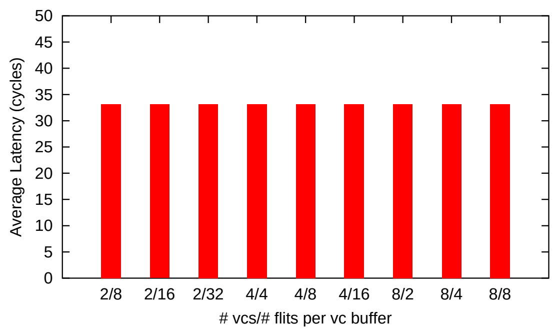


(c) 64 buffer slots per port

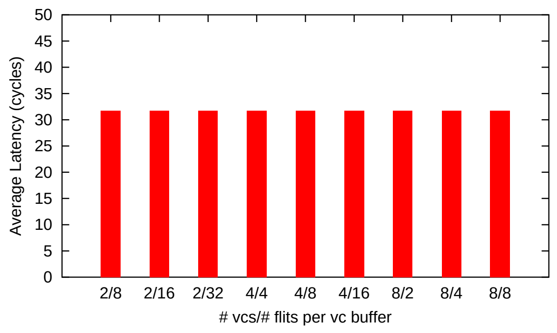
Figure 5.9: Latency vs. accepted traffic for 8x8 network with various amounts of buffer space.



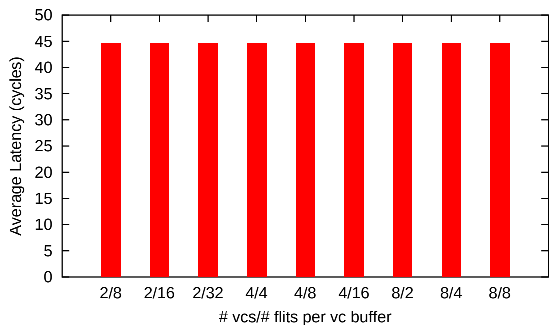
(a) FFT. Average message length: 15 flits.



(b) IFFT. Average message length: 16 flits.

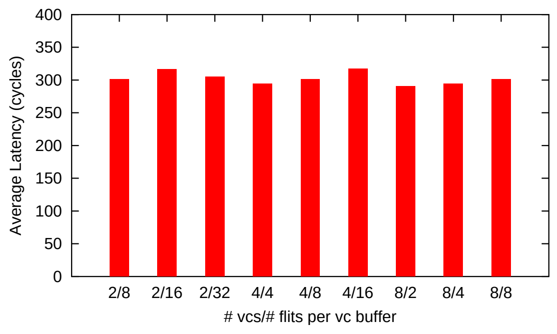


(c) Median Filter. Average message length: 14 flits.

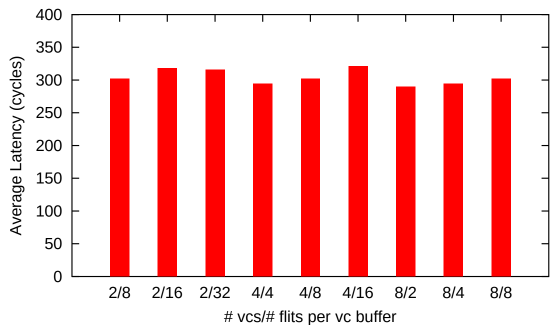


(d) Matrix Multiply. Average message length: 15 flits.

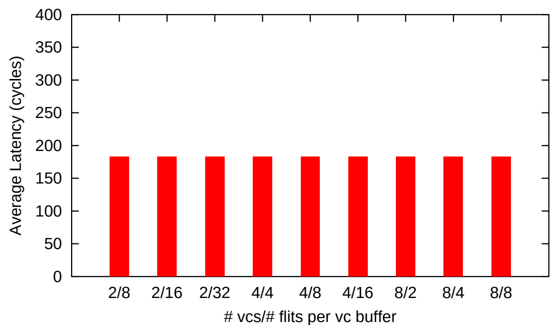
Figure 5.10: Thread message latency in application benchmarks for various 2x2 configurations.



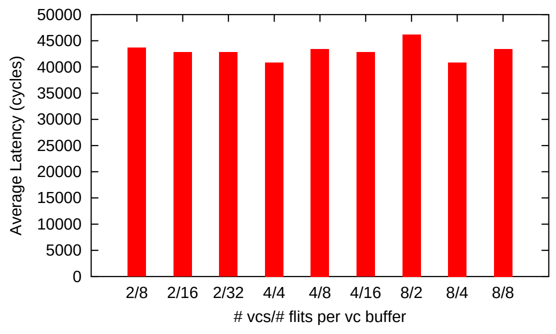
(a) FFT. Average message length: 76 flits.



(b) IFFT. Average message length: 76 flits.

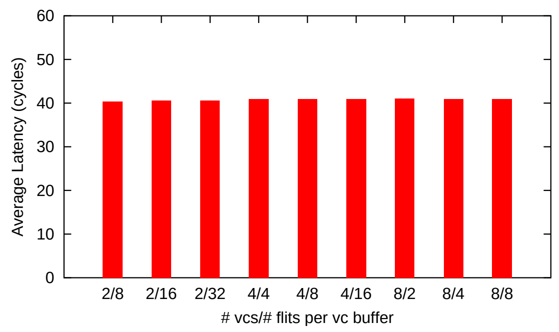


(c) Median Filter. Average message length: 46 flits.

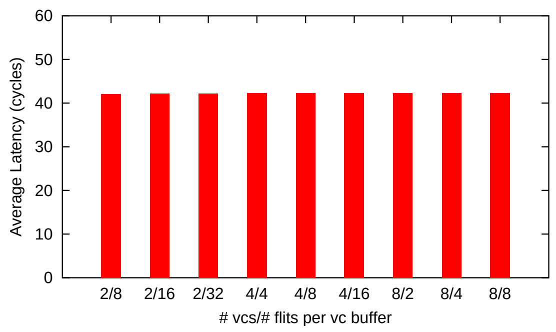


(d) Matrix Multiply. Average message length: 8203 flits.

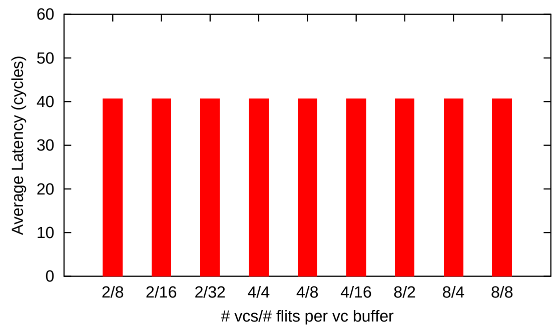
Figure 5.11: Data message latency in application benchmarks for various 2x2 configurations.



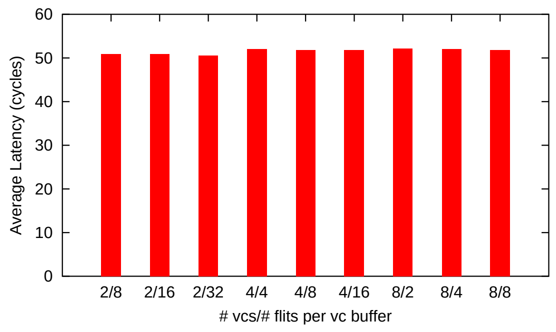
(a) FFT. Average message length: 24 flits.



(b) IFFT. Average message length: 25 flits.

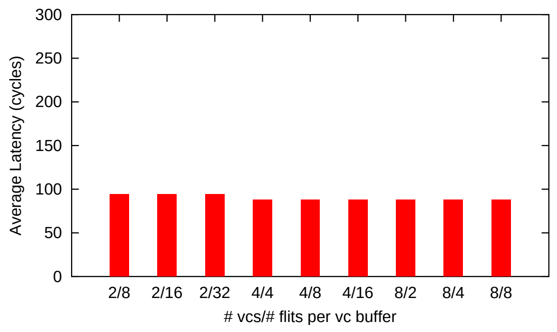


(c) Median Filter. Average message length: 18 flits.

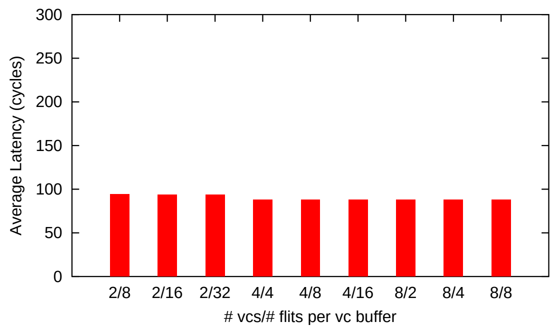


(d) Matrix Multiply. Average message length: 21 flits.

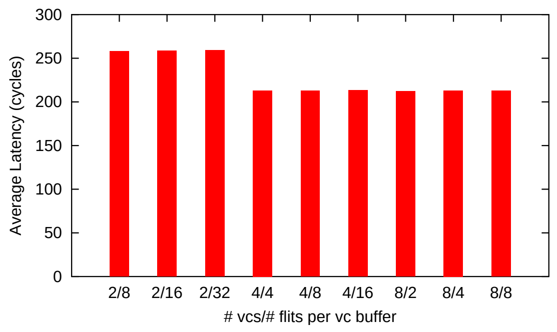
Figure 5.12: Thread message latency in application benchmarks for various 4x4 configurations.



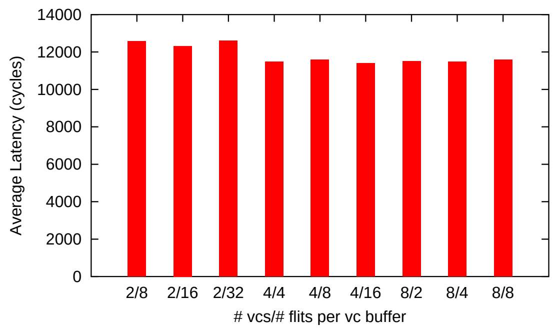
(a) FFT. Average message length: 35 flits.



(b) IFFT. Average message length: 35 flits.

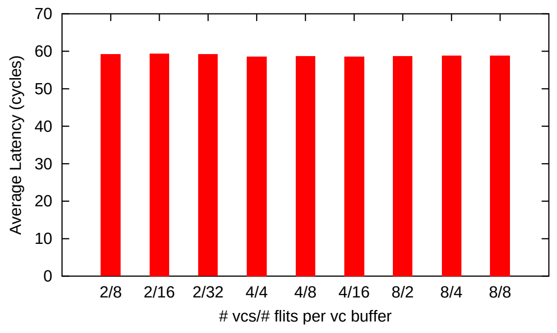


(c) Median Filter. Average message length: 48 flits.

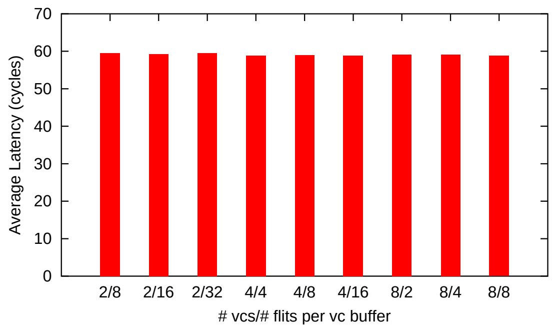


(d) Matrix Multiply. Average message length: 2063 flits.

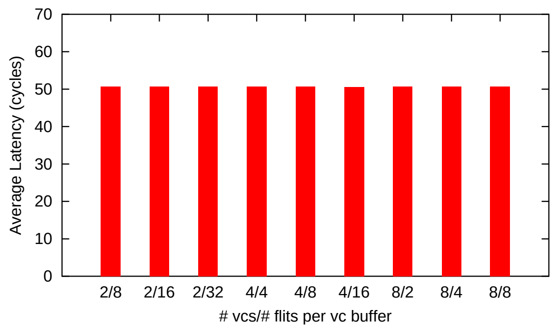
Figure 5.13: Data message latency in application benchmarks for various 4x4 configurations.



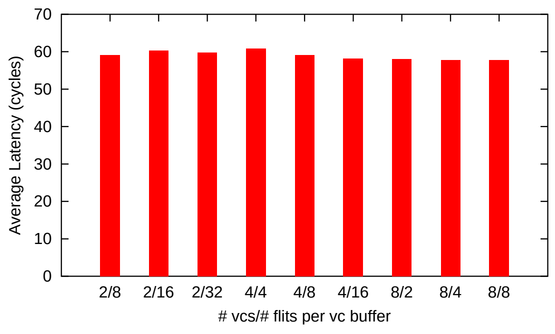
(a) FFT. Average message length: 40 flits.



(b) IFFT. Average message length: 40 flits.

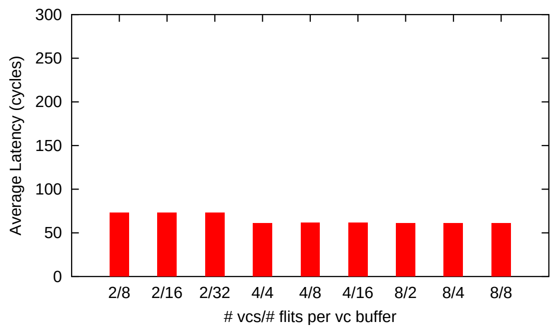


(c) Median Filter. Average message length: 27 flits.

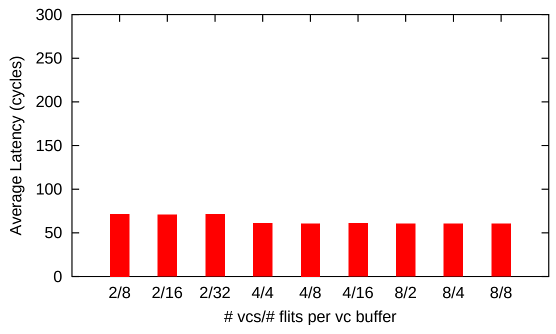


(d) Matrix Multiply. Average message length: 26 flits.

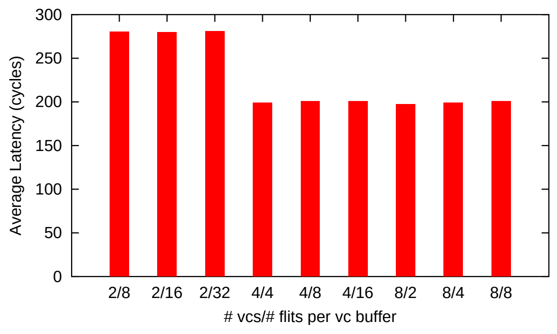
Figure 5.14: Thread message latency in application benchmarks for various 8x8 configurations.



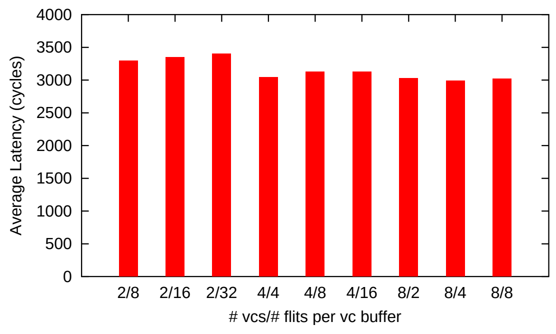
(a) FFT. Average message length: 40 flits.



(b) IFFT. Average message length: 37 flits.



(c) Median Filter. Average message length: 38 flits.



(d) Matrix Multiply. Average message length: 529 flits.

Figure 5.15: Data message latency in application benchmarks for various 8x8 configurations.

Chapter 6

Conclusions

6.1 Summary of Findings

In this thesis, a framework was developed for analyzing on-chip interconnection networks. The analysis consists of power, area, and performance, while considering implementation issues such as pipeline depth, stage delay, and crossbar size. The on-chip network in the SCMP parallel computer has been used as the case study for analyzing cost and performance tradeoffs.

In terms of area, full custom VLSI layout showed that flit buffers make up the majority of the area in the router component. A parametric area model was developed for the flit buffers and crossbar switch, allowing a reasonable estimate of area independent of fabrication technology.

Dynamic power consumption of the flit buffers and crossbar switch were estimated using architectural-level models of circuit capacitance. While the models are reusable in a variety of fabrication processes, the TSMC 0.18μ process was used as a sample of quantitative results in a modern sub-micron process. The results showed that power consumption of a single router is relatively small ($< 1W$), and primarily composed of dissipation in the flit buffers.

Performance tests were conducted in two components: synthetic network traffic and application benchmarks. The synthetic network traffic tested the router design exclusively, independent of external system factors. However, the application benchmarks ultimately test what matters most for users of the SCMP system: application performance. In both cases, larger system sizes had a greater reliance on the number of nodes, while smaller systems are dependent on buffer depth. For networks of 64 nodes, at least four virtual channels should be used. Smaller networks could likely use two virtual channels, but the flit buffers should be larger, with at least 32 flits of total storage per port.

Considering the power, area, and performance together, several conclusions can be drawn. First, virtual channels do not increase the power or area costs very much. While arbitration logic was not modeled in terms of area or power, other work has shown it to be significantly smaller than other components in the router. Second, increasing the depth of flit buffers without bound does not improve performance, but drastically impacts area and power. Careful consideration must be given when choosing the virtual channel configuration.

6.2 Future Work

While this thesis has made progress towards a framework for analyzing the cost and performance tradeoffs in network architecture, significant work remains. In terms of performance and simulation, the SCMP system needs a broader base of applications, including those utilizing a variety of traffic paradigms. In the longer term, an operating system for the SCMP system could likely stress the network in different ways than some of the applications used here.

Area and power models of the arbitration logic should be made for completeness, even though they are likely diminutive in comparison to the flit buffers or crossbar switch. The system-level layout of the complete router should be considered to determine how best to place the various components, and what area/power costs are incurred by the layout.

The power models need validation at several levels. While they are not intended to be perfectly accurate, the power models presented in [Wang03] have only been roughly validated. A more detailed layout of the flit buffers should be made, in particular to include the peripheral drive circuitry. The detailed buffer layout should be extracted, simulated, and refined to obtain accurate estimates of drive transistor size. Additionally, modern extraction tools are able to compute more accurate wire capacitance that includes coupling and wire-to-wire capacitance. These more accurate parameters should be included in the power models and comparisons must be made to determine the amount of modeling necessary for reasonably accurate estimates.

In the case of the larger SCMP systems, power dissipation will be a critical operating concern. With a validated power model for the router, similar efforts should be made for the other components of the SCMP system. Because of the similarity to conventional processing elements, the Wattch framework [Brooks00] and similar efforts can be used as a basis for modeling power consumption in SCMP. Because the main memory in SCMP will likely be implemented using embedded DRAM, models of eDRAM power consumption must be developed and validated as well.

As feature sizes shrink, threshold voltages decrease and leakage power increases. Within a few process generations, leakage power will dominate dynamic power, and must be considered as part of the power consumption picture [Powell01]. Very recent work has led to the development of leakage power models for routers [Chen03]. These models should be integrated with the dynamic power models presented here.

Finally, with the validation of the framework presented here, research ideas can be explored while keeping cost and performance in mind. This work has shown the cost implications of flit buffers, but has not considered other routing algorithms or flow-control implementations. A recent paper by Busch proposed routing without flow-control circuitry [Busch01], an idea that could dramatically reduce power and area consumption. For performance, however, the hot-potato routing Busch proposes may have to be combined with more conventional wormhole or circuit switching in a hybrid system.

Bibliography

[Agarwal00] V. Agarwal et al., “Clock Rate versus IPC: The End of the Road for Conventional Microprocessors.” *Proc. 27th Ann. Int’l Symp. Computer Architecture*, New York: ACM Press, 2000, pp. 248-259.

[Athas88] W. C. Athas and C. L. Seitz, “Multicomputers: Message-Passing Concurrent Computers.” *IEEE Computer*, vol. 21, no. 8, Aug. 1988, pp. 9-24.

[Baker02] J. M. Baker et al., “SCMP: A Single-Chip Message Passing Parallel Computer.” *Proc. Parallel and Distributed Processing Techniques and Applications, PDPTA’02*, CSREA Press, 2002, pp. 1485-1491.

[Boden95] N. Boden et al., “Myrinet: A Gigabit-per-Second Local Area Network.” *IEEE Micro*, vol. 15, no. 1, Feb. 1995, pp. 29-36.

[Brooks00] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations.” *Proc. 27th Int’l Symp. Computer Architecture*, June 2000, pp. 83-94.

[Busch01] C. Busch, M. Herlihy, and R. Wattenhofer, “Routing without Flow Control.” *Proc. 13th Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA 2001)*, July 2001, pp. 11-20.

[Chen03] X. Chen and L.-S. Peh, “Leakage Power Modeling and Optimization in Interconnection Networks.” To appear in *Proc. Int’l Symp. Low Power and Electronics Design*, Aug. 2003.

[Chien93] A. A. Chien, “A Cost and Speed Model for k-ary n-cube Wormhole Routers.” *Proc Hot Interconnects ’93*, Aug. 1993.

[Culler98] D. E. Culler and J. P. Singh, with A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, San Francisco: Morgan-Kaufmann, 1998.

[Dally90] W. J. Dally, “Performance analysis of k-ary n-cube interconnection networks.” *IEEE Trans. Comput.*, vol. 39, no. 6, June 1990, pp. 775-785.

[Dally92] W. J. Dally, “Virtual-Channel Flow Control.” *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 2, Mar. 1992, pp. 194-205.

[Dally98] W. J. Dally and J. W. Poulton, *Digital Systems Engineering.* Cambridge University Press, 1998.

[Dally01] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks." *Proc. DAC 2001*, New York: ACM Press, 2001, pp. 685-689.

[Diefendorff97] K. Diefendorff and P. K. Dubey, "How Multimedia Workloads Will Change Processor Design." *IEEE Computer*, vol. 30, no. 9, Sept. 1997, pp. 43-45.

[Duarte02] D. E. Duarte, N. Vijaykrishnan, and M. J. Irwin, "A clock power model to evaluate impact of architectural and technology optimizations." *IEEE Tran. VLSI*, vol. 10, no. 6, Dec. 2002, pp. 844-855.

[Duato02] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks*, 2nd ed., Los Alamitos, CA: IEEE Computer Society Press, 2002.

[vonEicken92] T. von Eicken et al., "Active Messages: a Mechanism for Integrated Communication and Computation." *Proc. 19th Ann. Int'l Symp. Computer Architecture*, New York: ACM Press, 1992, pp. 256-266.

[Fillo95] M. Fillo et al., "The M-Machine Multicomputer." *Proc. 28th Ann. Int'l Symp. Microarchitecture*, IEEE Press, 1995, pp. 146-156.

[Guerrier00] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections." *Proc. DATE 2000*, March 2000, pp. 250-256.

[Hinton01] G. Hinton et al., "The microarchitecture of the Pentium 4 Processor." *Intel Technology Journal*, February 2001.
`http://developer.intel.com/technology/itj/q12001/articles/art_2.htm`.

[Ho01] R. Ho, K. W. Mai, and M. A. Horowitz, "The Future of Wires." *Proc. IEEE*, vol. 89, no. 4, April 2001, pp. 490-504.

[HorowitzNotes] M. Horowitz, "Lecture 11: MOS Memory", from Stanford EE271 notes. Retrieved July 13, 2003 from http://www-classes.usc.edu/engr/ee-s/577bb/lect.11.pdf

[Hwang97] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, New York: McGraw-Hill, 1997.

[ITRS02] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors, 2002 Update*, 2002.

[James90] D. V. James et al., "Scalable Coherent Interface." *IEEE Computer*, vol. 23, no. 6, June 1990, pp. 74-77.

[Jantsch03] A. Jantsch and H. Tenhunen, *Networks On Chip*, Kluwer Academic Publishers, 2003.

[Kumar02] S. Kumar et al., "A network on chip architecture and design methodology." *Proc. Ann. Symp. VLSI*, Apr. 2002, pp. 105-112.

[Mukherjee02] S. S. Mukherjee et al., "The Alpha 21364 Network Architecture." *IEEE Micro*, vol. 22, no. 2, Jan.-Feb. 20002, pp. 26-35.

[Noakes93] M. Noakes, D. A. Wallach, and W. J. Dally, "The J-Machine Multicomputer: An Architectural Evaluation." *Proc. 20th Ann. Int'l Symp. Computer Architecture*, New York: ACM Press, 1993, pp. 224-235.

[Peh01] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers." *Proc. HPCA 2001*, Jan. 2001, pp. 255-266.

[Powell01] M. Powell et al., "Reducing leakage in a high-performance deep-submicron instruction cache." *IEEE Tran. VLSI*, vol. 9, no. 1, Feb. 2001, pp. 77-89.

[Seitz85] C. Seitz, "The Cosmic Cube." *Commun. ACM*, vol. 28, no. 1, pp. 22-33.

[Soininen03] J. Soininen et al., "Extending platform-based design to network on chip systems." *Proc. 16th Int'l Symp. VLSI Design*, Jan. 2003, pp. 401-408.

[Sutherland91] I. E. Sutherland and R. F. Sproull, "Logical effort: Designing for speed on the back of an envelope." *Proc. 13th Conf. Advanced Research in VLSI*, Mar. 1991, pp. 1-16.

[Taylor02] M. Taylor et al., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs." *IEEE Micro*, vol. 22, no. 2, Mar.-Apr. 2002, pp. 25-35.

[TSMC18] MOSIS, "MOSIS Parametric Test Results, Run: T28M, Technology: SCN018, Vendor: TSMC, Feature Size: 0.18 microns". Accessed July 13, 2003. `http://www.mosis.org/cgi-bin/cgiwrap/umosis/swp/params/ tsmc-018/t28m_lo_epi-params.txt`

[Waingold97] E. Waingold et al., "Baring it All To Software: RAW Machines." *IEEE Computer*, vol. 30, no. 9, Sept. 1997, pp. 86-93.

[Wang03] H.-S. Wang, L.-S. Peh, and S. Malik, "A power model for routers: modeling Alpha 21364 and InfiniBand routers." *IEEE Micro*, vol. 23, no. 1, Jan.-Feb. 2003, pp. 26-35.

[Weste94] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd ed. New York: Addison Wesley, 1994.

[Wills97] D. S. Wills et al., “High-Throughput, Low-Memory Applications on the Pica Architecture.” *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 10, Oct. 1997, pp. 1055-1067.

[Zyuban98] V. Zyuban, P. Kogge, “The energy complexity of register files.” *Proc. 1998 Int'l Symp. Low Power Electronics and Design*, Aug. 1998, pp. 305-310.

Vita

Brian Thomas Gold was born on July 6, 1979 in St. Charles, Missouri. In 1997 he graduated from Denbigh High School in Newport News, VA. After high school, he enrolled at Virginia Tech in Electrical Engineering and Applied Computational Mathematics. He received bachelor's degrees in both majors in Spring 2001.

Brian will complete the Master of Science degree in Summer 2003, sponsored by an NSF fellowship. In Fall 2003 he will begin the PhD program in Computer Engineering at Carnegie Mellon University in Pittsburgh, PA. He and his wife, Robin, now live in Pittsburgh.