High-Bandwidth Packet Switching on the Raw General-Purpose Architecture

by

Gleb Albertovich Chuvpilo

Submitted
to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2002

Author .....
Department of Electrical Engineering and Computer Science
August 20, 2002

Certified by .....
Saman Amarasinghe
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

High-Bandwidth Packet Switching on the Raw General-Purpose Architecture

by

Gleb Albertovich Chuvpilo

Submitted to the Department of Electrical Engineering and Computer Science
on August 20, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science
in Electrical Engineering and Computer Science

Abstract

One of the dictinct features of modern Internet routers is that most performance-critical tasks, such as the switching of packets, is currently done using Application Specific Integrated Circuits (ASICs) or custom-designed hardware. The only few cases when off-the-shelf general-purpose processors or specialized network processors are used are route lookup, Quality of Service (QoS), fabric scheduling, and alike, while existing general-purpose architectures have failed to give a useful interface to sufficient bandwidth to support high-bandwidth routing.

By using an architecture that is more general-purpose, routers can gain from economies of scale and increased flexibility compared to special-purpose hardware. The work presented in this thesis proposes the use of the Raw general-purpose processor as both a network processor and switch fabric for multigigabit routing. The Raw processor, through its tiled architecture and software-exposed on-chip networking, has enough internal and external bandwidth to deal with multigigabit routing.

This thesis has three main goals. First, it proposes a single-chip router design on the Raw general-purpose processor. We demonstrate that a 4-port edge router running on a 250 MHz Raw processor is able to switch 3.3 million packets per second at peak rate, which results in the throughput of 26.9 gigabits per second for 1,024-byte packets. Second, it shows that it is possible to obtain an efficient mapping of a dynamic communications pattern, such as the connections of the switch fabric of a router, to a compile-time static interconnect of the Raw processor tiles, and proposes a Rotating Crossbar design that achieves efficient routing on the Raw static network. Third, it proposes the incorporation of computation into the communication interconnect of the switch fabric of a router.

Thesis Supervisor: Saman Amarasinghe
Title: Associate Professor

Acknowledgments

First and foremost, I would like to thank my advisor, Saman Amarasinghe, for getting me started with this research and for his constant support and valuable advice. Saman is a great advisor, and it has been a great pleasure working with him throughout these two years. I am very lucky, and deeply grateful, to have met him.

My thanks to all of the members of the MIT Computer Architecture Group, especially Anant Agarwal, Michal Karczmarek, Diego Puppin, Bill Thies, David Wentzlaff, Michael Taylor, Walter Lee, Darko Marinov, Derek Bruening, Mark Stephenson, Sam Larsen, Michael Gordon, Chris Batten, Matthew Frank, Shireen Agah, and Cornelia Colyer – it has been very enjoyable to work with all of you.

I, nakonets, dorogiye moi mama i papa – spasibo vam ogromnoe za vsyu vashu podderzhku i lyubov', kotorye ya chuvstvoval vsegda i vezde, kak doma, tak i za tridevyat' zemel' ot nego. Vy – samoe dorogoe, chto u menia est'! Kak vy menia uchili s samogo detstva, glavnoe v zhizni – sledovat' printsipu "cherez ternii – k zvezdam!" i nikogda ne otstupat' pered trudnostyami. Vy okazalis' pravy – tak ono i est', i teper' ya ubedilsya v etom na sobstvennom opyte! Per aspera ad astra! Ya vas ochen' lyublyu! My vsegda budem vmeste!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The relentless growth of the Internet over the past few years has created a unique information space and provided us with fast and cheap means of communication. The rapid increase of available bandwidth was mainly instigated by the innovation of link technologies, especially the development of optical carriers, while as the routers that power the Internet have become a bottleneck in the rocketing use of the World Wide Web. With the advent of gigabit networking [18], sophisticated new distributed router designs have emerged to meet the resulting technical challenges in ways that allow Internet Service Providers (ISPs) to quickly scale up their networks and bring new services to market. [3]

One of the dictinct features of modern Internet routers is that most performance-critical tasks, such as the switching of packets, is currently done using Application Specific Integrated Circuits (ASICs) or custom-designed hardware. The only few cases when off-the-shelf general-purpose processors or specialized network processors are used are route lookup, Quality of Service (QoS), fabric scheduling, and alike, while existing general-purpose architectures have failed to give a useful interface to sufficient bandwidth to support high-bandwidth routing.

By using an architecture that is more general-purpose, routers can gain from economies of scale and increased flexibility compared to special-purpose hardware. The work presented in this thesis proposes the use of the Raw general-purpose processor [21] as both a network processor and switch fabric for multigigabit routing. The

Raw processor, through its tiled architecture and software-exposed on-chip networking, has enough internal and external bandwidth to deal with multigigabit routing.

1.1 Problem Statement

This thesis has three main goals. First, it proposes a single-chip router design on the Raw general-purpose processor. We demonstrate that a 4-port edge router running on a 250 MHz Raw processor is able to switch 3.3 million packets per second (Mpps) at peak rate, which results in the throughput of 26.9 gigabits per second (Gbps) for 1,024-byte packets. Second, it shows that it is possible to obtain an efficient mapping of a dynamic communication pattern, such as the connections of the switch fabric of a router, to a compile-time static interconnect of the Raw processor tiles, and proposes a Rotating Crossbar design that achieves efficient routing on the Raw static network. Third, it proposes the incorporation of computation into the communication interconnect of the switch fabric of a router. The addition of computation is motivated by two reasons: a growing need for routers to operate on the data payload of packets to provide extended services, such as encryption, and the fact that the addition of computation to the switch fabric removes the difficulty of bringing data near to a computational resource that is able to compute on it.

The contributions of this thesis include:

- A router design on the Raw general-purpose processor that achieves 26.9 Gbps performance;
- A Rotating Crossbar design as an efficient mapping of a dynamic communication pattern to a compile-time static interconnect;
- A distributed scheduling algorithm for the Rotating Crossbar;
- A minimization procedure on the configuration space of the Rotating Crossbar.

1.2 Thesis Overview

The roadmap to the thesis looks as follows: Chapter 2 describes the foundations of routing and examines typical architectures of existing Internet routers. In this chapter we will take a look at a case study of an MGR router from BBN, a Cisco 12000 Gigabit Switch Router backplane, and a network processor called IXP1200 manufactured by Intel. Chapter 3 then describes the Raw general-purpose processor on which our router is built, including its Instruction Set Architecture, communication mechanisms, and performance.

The next chapters examine the Raw Router architecture and a complete router configuration. Chapter 4 presents the chosen partitioning of the Raw processor, the path that the packets take through the router, and other general issues, such as buffer management. Chapter 5 moves from general descriptions to specifics, describing the design of the router's switch fabric and the Rotating Crossbar algorithm. Several sections show the properties of this algorithm, including fairness and absence of possible deadlocks.

Chapter 6 introduces a distributed scheduling algorithm for the Rotating Crossbar, and explains how the constraints on the memory system of the Raw processor influence on the implementation, and show a minimization of the configuration space made in order to fit the code in a tile's local instruction memory. This chapter also describes the timing of the algorithm at run-time, as well as the programming techniques used on the Crossbar Processors.

Chapter 7 describes the results of our work – the peak and aggregate performance of the Raw Router compared to the Click router, which is another router implemented on a general-purpose processor. The chapter shows that we have achieved the goal of building a multigigabit router on Raw. This chapter also studies the efficiency of the current implementation and explains the utilization of the Raw processor on a per-tile basis. The analysis also suggests a general approach to obtain the maximum utilization of the router.

Chapter 8 decribes the future improvements that we are planning to add to the

existing router, including new designs pursuing full utilization of the Raw processor, the implementation of the IP route lookup on Raw, the issues of scalability and support of multicast traffic in the switch fabric, flow prioritization to deploy Quality of Service, as well as the application of the current router layout for routing in low earch orbit satellite systems.

Finally, Chapter 9 concludes the thesis.

Chapter 2

An Overview of the Architecture of Internet Routers

This chapter describes the foundations of routing and examines typical architectures of existing Internet routers. In this chapter we will take a look at a case study of an MGR router from BBN, a Cisco 12000 Gigabit Switch Router backplane, and a network processor called IXP1200 manufactured by Intel. The point that we would like to make is that most performance-critical tasks, such as the switching of packets in the fabric connecting different network ports of a modern high-performance router is currently done using ASICs or custom-designed hardware. The only few cases when off-the-shelf general-purpose processors or specialized network processors are used are route lookup, Quality of Service (QoS), fabric scheduling, and alike, while existing general-purpose architectures have failed to give a useful interface to sufficient bandwidth to support multigigabit routing.

2.1 Router Basics

A high-performance router generally consist of four major components shown in Figure 2-1: a network processor, a set of forwarding engines, a set of interfaces, and a switch fabric. [3]

The **Network Processor** is used to calculate the best path from packet source

to destination. The knowledge about best paths becomes available through sharing information about network conditions with neighboring routers. In most cases route processing is centralized to reduce system complexity, because the timing of routing table updates is independent of packet forwarding. Common routing protocols that network processors implement are Border Gateway Protocol (BGP), Open Shortest Path First (OSPF), and Routing Information Protocol (RIP).

The next element is a set of **Forwarding Engines**. The forwarding engines are responsible for deciding to which output line card to forward each packet. To do that, a forwarding engine consults a local copy of the routing table, which is a summary of all routing tables in the system. This database contains the mapping of destination IP addresses to output interface numbers where the packets with these destinations should be forwarded. The amount of route lookups is proportional to the aggregate traffic serviced by a router, which is why frequently forwarding engines are designed in a distributed manner. Traditional implementations of routing tables use a version of Patricia trees [15] with modifications for longest prefix matching.

The next component is a set of **Interfaces** with respective **Memory Systems** typically located on interfaces. The memory system buffers incoming packets to absorb bursts and temporary congestion, both of which result from the use of the bursty Transmission Control Protocol (TCP). The buffer space is needed as a temporary waiting area where packets queue up for transmission when there is contention for output interfaces. The memory system of an interface must be prepared to buffer up to bandwidth $\times$ delay worth of data that is said to be "in flight" in a pipe between a sender and a receiver. Thus, for each gigabit of bandwidth, assuming a cross-country round-trip time of 100 ms, an interface requires 12.5 megabytes of buffer space.

Finally, all of the components have a common rendez-vous point, which is called the **Switch Fabric**. The function of the switch fabric is to allow the transmission of packets from incoming interfaces to outgoing interfaces, as well as enable interfaces to query route information from forwarding engines and permit the network processor to update local copies of the routing table on each of the forwardng engines.

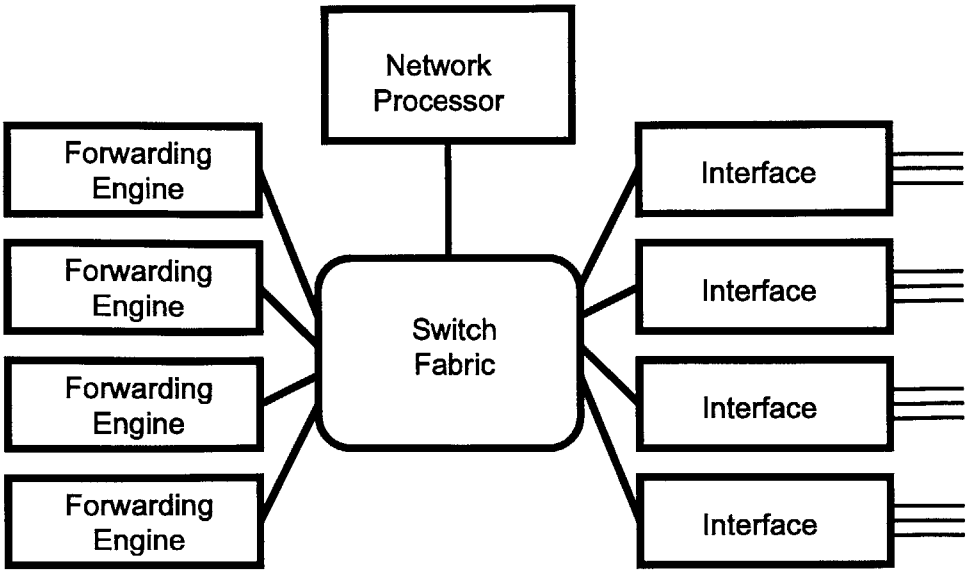


Figure 2-1: A typical router design.

2.2 Case Study

As we mentioned earlier, here we will examine and compare typical architectures of existing Internet routers: an MGR router made by BBN, the backplane of a Cisco 12000 Gigabit Switch Router, and IXP1200 – a network processor from Intel. They all exemplify successful engineering efforts in making high-performance routers. The MGR router was one earliest multigigabit routers built, and it was a research effort based on the Butterfly switching matrix, funded by ARPA, with Craig Partridge was involved. MGR forwards up to 32 million packets per second depending on configuration. The Cisco 12000 Gigabit Switch Router is a crosspoint switch and router available with 4, 8 and 12 slots. It supports IP over SONET as well as ATM and connects directly into fiber infrastructure with OC-3 or OC-12 speeds, later on OC-48. During development it was known as the BFR (Big Fast Router). Finally, IXP1200 and its successor Castine are the flagship network processors of the Intel Internet Exchange Architecture.

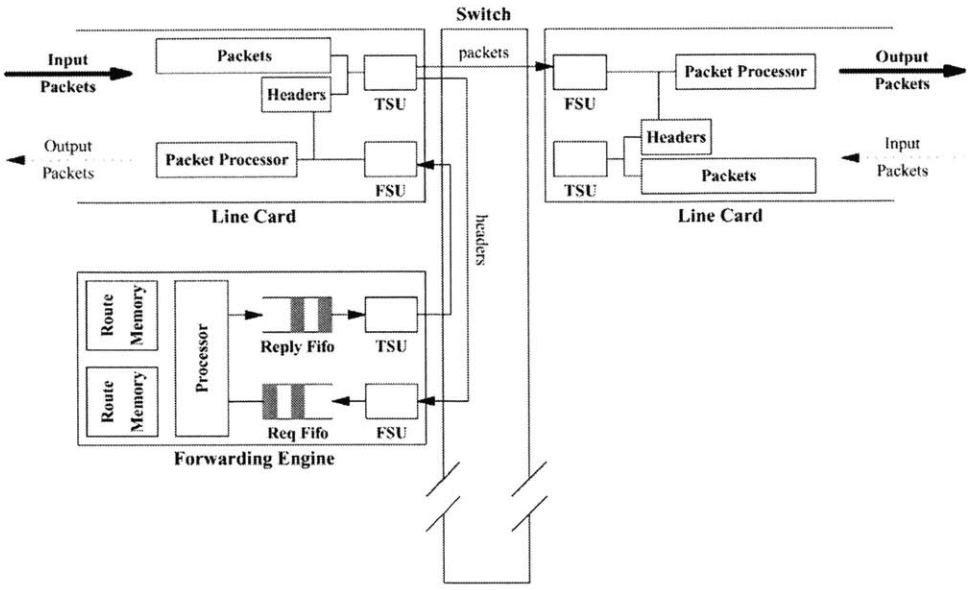


Figure 2-2: An outline of the MGR router (Adapted from [17]).

2.2.1 Router Layout: An MGR Router from BBN

This section describes a router designed by BBN and called MGR [17]. This router achieves up to 32 million packet per second forwarding rates with 50 Gbps of full-duplex backplane capacity.

An outline of the MGR router is shown in Figure 2-2. It consists of multiple line cards each supporting one or more network interfaces, and forwarding engine cards, all plugged into a high speed switch. When a packet arrives at a line card, its header is removed and passed through the switch to a forwarding engine. The remainder of the packet remains on the inbound line card. The forwarding engine reads the header to determine how to forward the packet and then updates the header and sends the updated header and its forwarding instructions back to the inbound line card. The inbound line card integrates the new header with the rest of the packet, and sends the entire packet to the outbound line card for transmission. Not shown in Figure 2-2 but an important piece of the router is a network processor that provides basic management functions such as link up/down management and generation of forwarding engine routing tables for the router.

The Forwarding Engines. When a line card receives a new packet, it sends the

packet header to a forwarding engine. The forwarding engine then determines how the packet should be routed. At the heart of each forwarding engine is a 415 MHz Digital Equipment Corporation Alpha 21164 processor, which is a 64-bit, 32-register, super-scalar RISC processor.

The Switch. The MGR uses a 15-port switch to move data between function cards. The switch is a point-to-point switch. The major limitation to a point-to-point switch is that it does not support the one-to-many transfers required for multicasting. Multicast packets in the MGR router are copied multiple times, once to each outbound line card. The MGR switch is an input-queued switch in which each input keeps a separate FIFO and bids separately for each output. Keeping track of traffic for each output separately means the switch does not suffer Head-of-Line blocking and it has been shown by simulation and more recently proven that such a switch can achieve 100% throughput. The key design choice in this style of switch is its allocation algorithm – how one arbitrates among the various bids. The MGR arbitration seeks to maximize throughput, at the expense of predictable latency.

Line Card Design. A line card in the MGR can have up to sixteen interfaces on it. However, the total bandwidth of all interfaces on a single card should not exceed approximately 2.5 Gbps. The difference between the 2.5 Gbps and the 3.3 Gbps switch rate is to provide enough switch bandwidth to transfer packet headers to and from the forwarding engines. The 2.5 Gbps rate is sufficient to support one OC-48c (2.4 Gbps) SONET interface, four OC-12c (622 Mbps) SONET interfaces or three HIPPI (800 Mbps) interfaces on one card. It is also more than enough to support sixteen 100 Mbps Ethernet or FDDI interfaces.

The Network Processor. The network processor is a commercial PC motherboard with a PCI interface. This motherboard uses a 21064 Alpha processor clocked at 233 MHz. The Alpha processor was chosen for ease of compatibility with the forwarding engines. The motherboard is attached to a PCI bridge which gives it access to all function cards and also to a set of registers on the switch allocator board. The processor runs the 1.1 NetBSD release of UNIX, which is a freely available version of UNIX based on the 4.4 BSD software release.

Managing Routing and Forwarding Tables. Routing information in the MGR is managed jointly by the network processor and the forwarding engines. All routing protocols are implemented on the network processor, which is responsible for keeping complete routing information. From its routing information, the network processor builds a forwarding table for each forwarding engine. These forwarding tables may be all the same, or there may be different forwarding tables for different forwarding engines. One advantage of having the network processor build the tables is that while the network processor needs complete routing information such as hop counts and who each route was learned from, the tables for the forwarding engines need simply indicate the next hop. As a result, the forwarding tables for the forwarding engines are much smaller than the routing table maintained by the network processor.

2.2.2 Switch Fabric: Cisco 12000 GSR Backplane

In this section we will focus on the switched backplane developed for the Cisco 12000 Series Gigabit Switch Routers (GSR) [12, 4, 3]. This router has a high-performance switched backplane architecture capable of switching 16 ports simultaneously, each with a line rate of 2.4 Gbps. The backplane uses a number of new technologies that enable a parallel, compact design providing high throughput for both unicast and multicast traffic. Integrated support for priorities on the backplane allows the router to provide distinguished qualities of service for multimedia applications. Figure 2-3 shows the structure of a typical crossbar switch.

Why Fixed Length Packets. Packets may be transferred across the switched backplane in small fixed sized units, or as variable length packets. There are significant disadvantages against using variable length packets, and so the highest-performance routers segment variable length packets into fixed sized units, or "cells", before transferring them across the backplane. The cells are reassembled back into variable length packets at the output before being transmitted on the outgoing line. Let's examine the choice between using fixed, and variable length packets, as shown in [12].

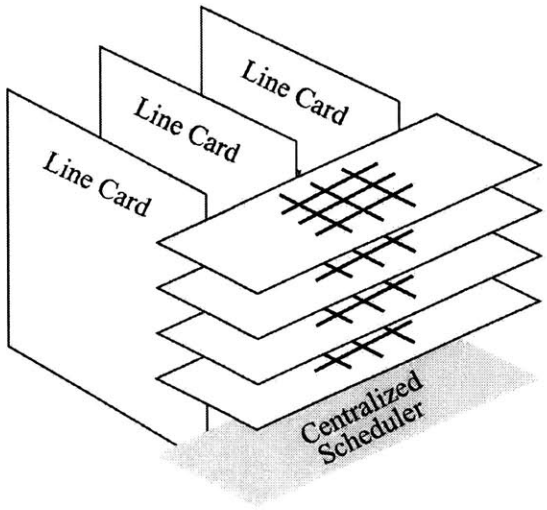


Figure 2-3: A four-way parallel crossbar switch, interconnecting multiple line cards. A centralized scheduler connects to each line card, and determines the configuration of the crossbar switch for each time slot (Adapted from [12]).

There are no problems if we use fixed size cells – the timing of the switch fabric is just a sequence of fixed size time slots. The scheduling algorithm allocates all of the resources of the switch fabric at the beginning of every time slot, and there is no need to keep track of when each and every data transfer ends, which make the hardware simple and faste. However, with variable length packets things are getting much more complicated, especially for the scheduler – it must do a lot of bookkeeping to keep track of available and unavailable outputs. It often needs to decide whether to allocate an idle output or wait for a busy one to become free, in order to both minimize starvation and maximize aggregate throughput.

It is shown that using fixed length packets ("cells") allows up to 100% of the switch bandwidth to be used for transferring cells. If variable lentgh packets are used, the system throughput is limited to approximately 60%. [12]

Why Virtual Output Queueing. Even though a crossbar switch is always internally non-blocking, there are three other types of blocking that can limit its performance [12]. The first type of blocking is called head-of-line (HOL) blocking; the other two are input-blocking and output-blocking. HOL-blocking can significantly re-

duce the performance of a crossbar switch, wasting approximately 40% of the switch bandwidth . Fortunately, there is a solution for this problem called virtual output queueing, that eliminates HOL-blocking entirely and makes 100% of the switch bandwidth available for transferring packets.

The other types of blocking, input- and output-blocking, are present in all crossbar switches. They arise because of contention for access to the crossbar: each input line and each output line of a crossbar can only transport one cell at a time. If multiple cells wish to access a line simultaneously, only one will gain access while the others will be queued. Input- and output-blocking do not reduce the throughput of a crossbar switch. Instead, they increase the delay of individual packets through the system, and perhaps more importantly make the delay random and unpredictable.

There is a simple fix to the HOL-blocking problem known as virtual output queueing (VOQ), first proposed in [19]. At each input, a separate FIFO queue is maintained for each output. After a forwarding decision has been made, an arriving cell is placed in the queue corresponding to its outgoing port. At the beginning of each time slot, a centralized scheduling algorithm examines the contents of all the input queues, and finds a conflict-free match between inputs and outputs.

It is shown that if VOQs are used, instead of the conventional FIFO queues, then HOL blocking can be eliminated entirely. This raises the system throughput from 60% to 100%. [12]

Crossbar Scheduling Algorithm. The Cisco 12000 GSR Backplane uses the iSLIP scheduling algorithm [13],which attempts to quickly converge on a conflict-free match in multiple iterations, where each iteration consists of three steps. All inputs and outputs are initially unmatched and only those inputs and outputs not matched at the end of one iteration are eligible for matching in the next. The three steps of each iteration operate in parallel on each output and input. The steps of each iteration are:

1. **Request.** Each input sends a request to every output for which it has a queued cell.

2. **Grant.** If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted.

3. **Accept.** If an input receives a grant, it accepts the one that appears next in a fixed, roundrobin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the accepted output. The pointer to the highest priority element at the corresponding output is incremented (modulo N) to one location beyond the granted input. The pointers are only updated after the first iteration.

By considering only unmatched inputs and outputs, each iteration matches inputs and outputs that were not matched during earlier iterations. It is shown that if a single scheduler is used to schedule both unicast and multicast traffic, then: (a) the scheduling decisions can be made at greater speed, and (b) the relative priority of unciast and multicast traffic can be maintained, preventing either type of traffic from starving the other. Also, if multicast traffic is queued separately, then the crossbar may be used to replicate cells, rather than wasting precious memory bandwidth at the input, and if the crossbar implements fanout-splitting for multicast packets, then the system throughput can be increased by 40%. [12]

2.2.3 Network Processor: Intel IXP1200

In this section we will take a look at the current flagship network processors of the Intel Internet Exchange Architecture – IXP1200 and Castine [1] [1]. Intel IXP1200 is shown in Figure 2-4. This processor runs at 232 MHz and forwards packets at the rate of 3.5 Mpps. It has a 2K control store in each of the 6 microengines, each of which is a RISC microprocessor. In order to keep the microengine pipeline busy, the microprocessor needs to run in a multi-threading mode.

[1]The material of this section is taken from Matthew Adiletta's talk on Intel network processors given at the MIT Laboratory for Computer Science, April 2002

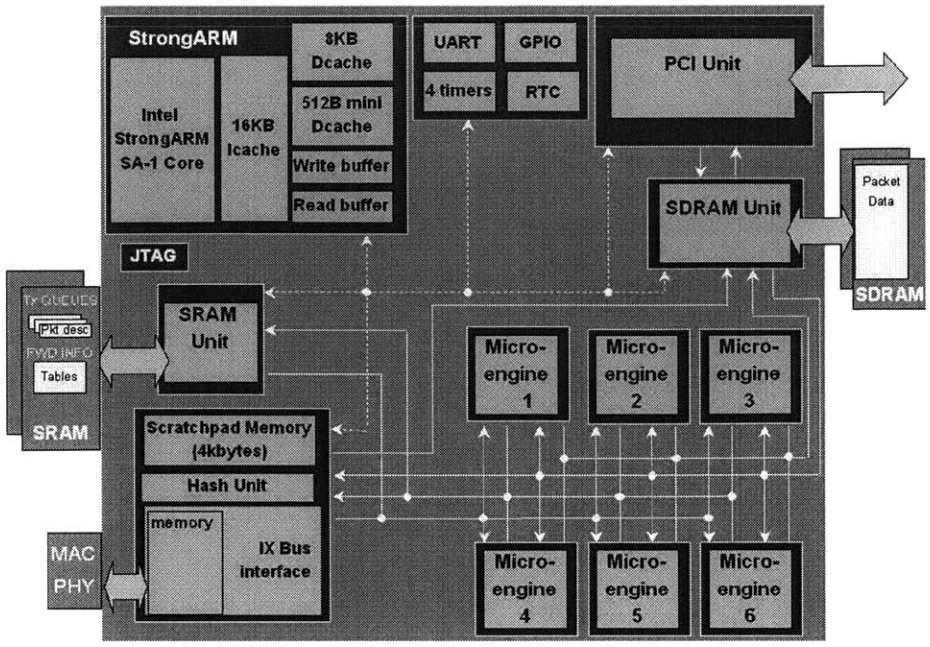


Figure 2-4: Intel IXP1200 network processor (Adapted from [1]).

Intel Castine will be a 3rd Generation Network Processor to come out from Intel. It will have 16 microengines, each running at 1.4 GHz (over 23,100 Mips). It will have an Integrated 700 MHz XScale Control Plane Processor, a 10+ Gbps full duplex Media Interface, 50 Gbps packet memory bandwidth, 30 million packets per second L4 forwarding, 60 million enqueue/dequeue operations per second, and advanced hardware support for queueing, traffic shaping, SARing, and policing.

2.3 Why Not Use an ASIC?

The MGR uses an off-the-shelf processor to make forwarding decisions, which is often the case for many contemporary Internet routers. One can ask whether there is any reason not to use an ASIC for that purpose, and implement a forwarding engine and a network processor in a more cost effective way. Indeed, the IPv4 specification has been around for quite a while, and it is very stable. So what is a problem with an ASIC?

The answer to this question depends on the specific location where the router might be deployed. If a router is targeted at a corporate LAN, then an ASIC may be a good solution. However, ISPs often require their equipment to be flexible and easily upgradable – for instance, ready to work with IPv6 – which is why a programmable non-ASIC approach in building IP routers wins.

2.4 Software Routers on General-Purpose Processors

Another approach was explored in the Click Router [14, 10]. The idea was to build a software router running on a general-purpose architecture that would be flexible, configurable, and cheap. Unfortunately, conventional general-purpose processors do not provide enough of input/output bandwidth to carry out multigigabit routing, which is why most fast routers nowadays work on special-purpose processors.

Chapter 3

A Brief Description of the Raw processor

This chapter describes the Raw general-purpose processor on which our router is built, including its Instruction Set Architecture, communication mechanisms, and performance. The Raw processor is a general purpose processor designed to take advantage of Moore's Law – the availability of large quantities of fast transistors.

3.1 Processor Layout

The general organization of the Raw Processor (Figure 3-1) is as a chip multiprocessor with multiple fine grain, first class, register mapped communication networks. The processor contains 16 tiles in a 4×4 mesh grid. A tile consists of a tile processor, memory, two dynamic network routers, two static switch crossbars and a static switch processor. Tiles are connected to each of their four nearest neighbors by two sets of static network interconnect and two sets of dynamic network interconnect. The Raw instruction set architecture works together with this parallel architecture by exposing both the computational and communication resources up to the software. By exposing the communication delays up to the software, compilers can do better jobs at compiling because they are able to explicitly manage wire delay and spatially map computation appropriately. This is in sharp contrast to approaches of other instruc-

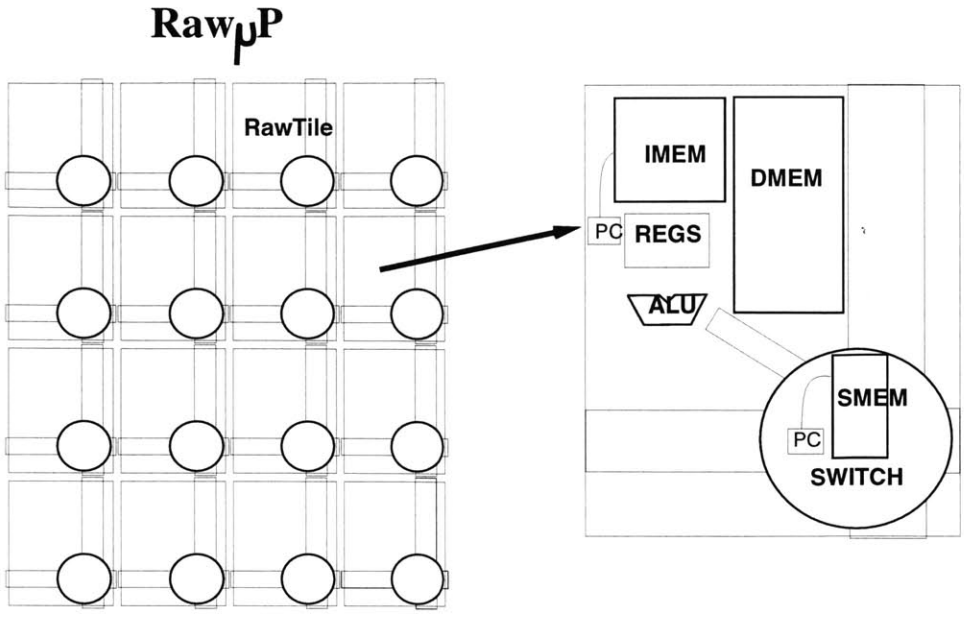


Figure 3-1: The Raw Processor with 16 tile processors in a 4×4 grid.

tion sets, which effectively mask wire delay. Because communication delay is exposed up to the software, this allows for larger scaling of functional units where conventional superscalar processors would break down because these wire delays would exist, but would have no way to be managed by software. The Raw project is examining larger configurations and hence Raw Processors can be seamlessly connected to build fabrics of up to 1,024 tiles.

3.2 Raw Instruction Set Architecture

Each Raw chip has 16 tile processors, one in each tile. A tile processor is a 32-bit 8-stage pipelined MIPS-like processor. Each tile processor contains a fully pipelined two-stage integer multiplier, and a pipelined four-stage single precision floating point unit. The tile processor's instruction set is roughly equivalent to that of a R4000 with a few additions for communication applications, such as bit level extraction, masking and population related operations. The tile processor uses static branch prediction instead of delay slots. It has no branch penalty for properly predicted branches and a three-cycle penalty for mispredicted branches. The tile processor is also tightly

integrated with its corresponding communication resources. Each network is directly mapped into the register space. Network registers can be used as both a source and destination for instructions.

The Raw processor has 2,048 kilobytes of SRAM on-chip, with each tile having 8,192 words (32-bit word) of local instruction memory, and 8,192 words (64-bit word) of switch memory. A tile has a 8,192 word (32-bit word), 2-way set-associative, 3 cycle latency data cache with 32-byte lines. The cache uses a two-element bypassing write buffer to defer stores until after the tag has been checked. The memory can be in two modes, cached and uncached, and it can do 16 parallel accesses, one access per tile. There is no cache coherence support in the Raw processor.

3.3 Communication Mechanisms

The main communication mechanism in the Raw Processor is the static switch network. The code sequence shown in Figure 3-2 takes five cycles to execute. In the first cycle, tile 0 executes the OR instruction, and the value arrives at switch 0. On the second cycle, switch 0 transmits the value to switch 4. On the third cycle, switch 4 transmits the value to the processor. On the fourth cycle, the value enters the decode stage of the processor. On the fifth cycle, the AND instruction is executed. Since two of those cycles were spent performing useful computation, the send-to-use latency is three cycles.

The static network is controlled by a simple six-stage switch processor which configures a tile's static network crossbar on a per cycle basis. The Raw static network is flow-controlled and stalls when data is not available. The static network relies on compile time knowledge so that it can be programmed with appropriate control instructions and routes. The name static network is somewhat of a misnomer because it is very programmable. The static switch network has a completely independent instruction stream and is able to take simple branches. Thus, it is very well suited for compile-time known communication patterns and is able to handle these without the need for headers, which are found in dynamic networks. Each tile contains one switch

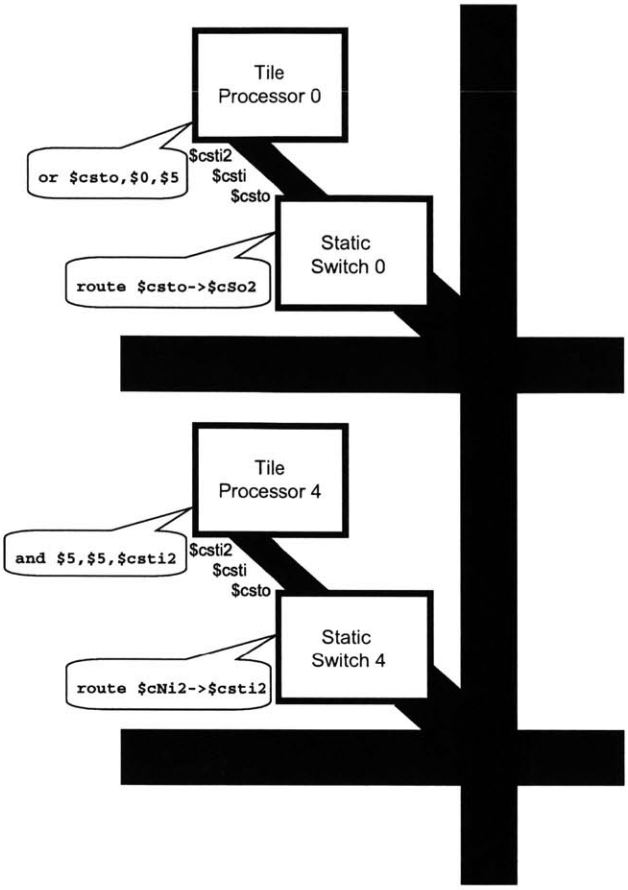


Figure 3-2: The switch and tile code required for a tile-to-tile send to the South from tile 0 to tile 4.

processor but two switch networks. The one switch processor can control the crossbar on each of five directions (North, South, East, West, and into the tile processor) for each switch network independently.

The dynamic networks on Raw are there to assist communication patterns that cannot be determined easily at compile time. Examples of this are external asynchronous interrupts and cache misses. Each tile has two identical dynamic networks. The dynamic network is a wormhole routed [5], two-stage pipelined, dimension-ordered network. The dynamic network uses header words to dynamically route messages on a two-dimensional mesh network. Messages on this network can vary in length from only the header up to 32 words including the header. Nearest neighbor ALU-to-ALU communication on the dynamic network takes between 15 and 30

cycles.

3.4 Performance

The Raw Processor Prototype is fabricated on IBM's SA-27E, 6 layer metal copper 0.15μ process. The Raw Processor is expected to operate at 250 MHz, and have 3.6 GOPS/GFLOPS of peak performance. It has 230 Gbps of bisection bandwidth and 201 Gbps of external chip bandwidth. Access to this off-chip bandwidth is provided through the Raw Processor's networks. To connect off-chip, the native internal networks are multiplexed through 1080 signal input/output pins. More information on the Raw microarchitecture can be found in the Raw Processor Specification. [20]

Chapter 4

An Overview of the Raw Router Architecture

This and the following chapters examine the Raw Router architecture and a complete router configuration. This chapter presents the chosen partitioning of the Raw processor, the path that the packets take through the router, and other general issues, such as buffer management.

4.1 Research Goals

The goal of this research was to design a multigigabit single-chip router solution using the Raw Processor and devise a switching algorithm for it. Some assumptions and practical considerations have influenced the design of this router. First of all, the goal of this design was to build an edge router or a scalable switch fabric of a core router, but not a complete core router. Many of the ideas presented here can be leveraged to build core routers, but considerations, such as limited internal buffer space and complex IP routing lookups require more analysis. Another design point is that this design is for a 4-input and 4-output router, and larger configurations are still to be explored in the future.

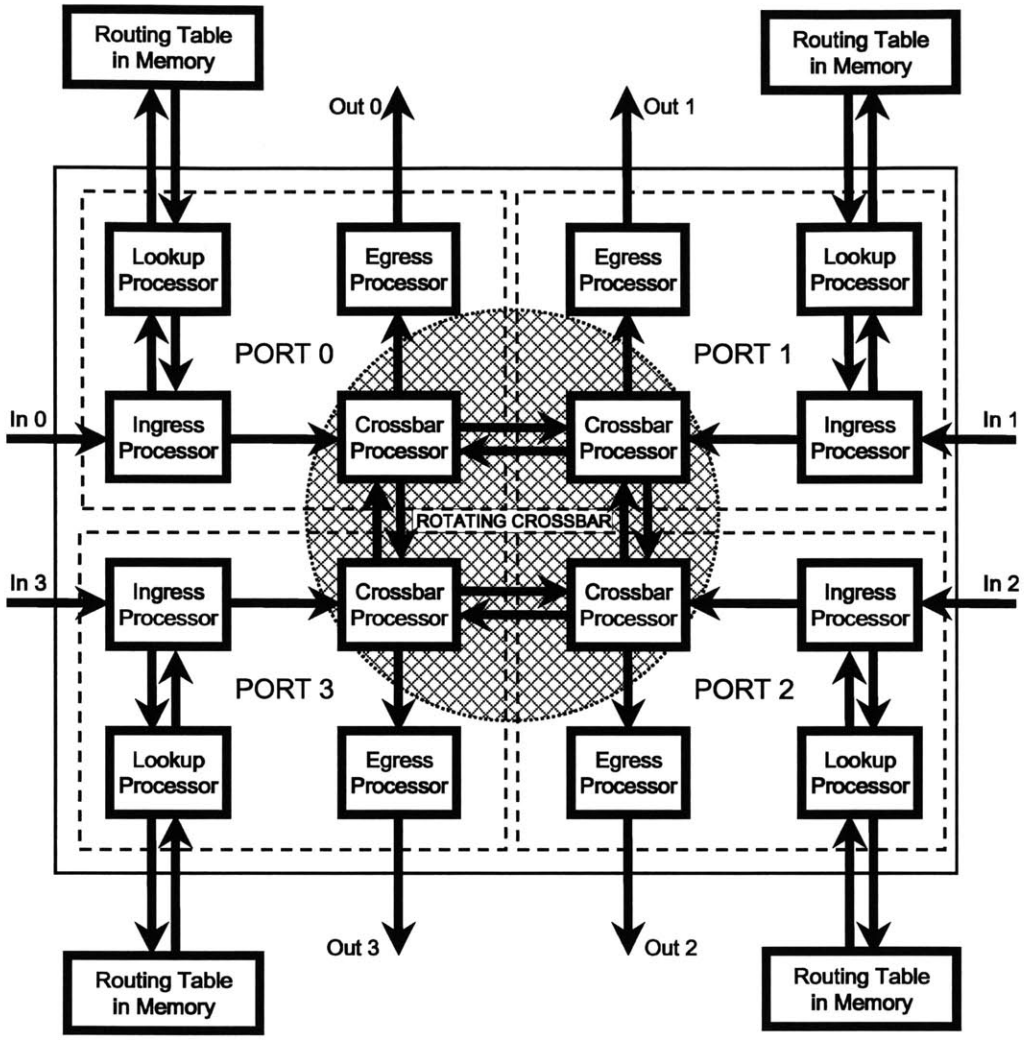


Figure 4-1: Mapping router functional elements to Raw tiles. Each of the four ports is comprised of four elements: an Ingress Processor, a Lookup Processor, a Crossbar Processor, and an Egress Processor.

4.2 Partitioning of the Raw Processor

The ability to carry out complex communication patterns quickly and efficiently is critical to implement a high-bandwidth router. The ability to statically orchestrate the computation and communication on the Raw processor's software-exposed parallel tiles and software-controlled static communication networks makes this general-purpose processor well suited for such an implementation. Thus, the first task in designing a router on Raw is to partition the router components and map them on to the Raw tiles. This mapping should balance the computation load between the 16 tile processors of Raw. More importantly, the mapping has to efficiently support the communication patterns of the router.

Figure 4-1 shows graphically the mapping that was chosen. Each of the four ports uses four tiles. An **Ingress Processor** is used to stream in and buffer data coming from the line card, as well as to perform the necessary processing of the IP header, including the checksum computation and decrement of the "Time to Live" field. This tile is also used for fragmentation of IP packets if their size exceeds the internal tile-to-tile data transfer block on the Raw chip. A **Lookup Processor** is necessary for accessing the routing table in the off-chip memory. **Crossbar Processors** form a **Rotating Crossbar** and they are utilized to transfer data between ports. An **Egress Processor** is used to perform the reassembly of large IP packets fragmented by the Ingress Processor, babysit the output line, and stream data to the output line card. The architecture of the Raw processor lends itself to straightforward replication of the port four times resulting in a 4×4 IP router.

4.3 Data Path Logistics

The path that data travels through this router is as follows. First data streams in on the static network from an off-chip input line card. The IP header, but not the data payload, of this packet is sent over the static network to the Lookup Processor to do classification and route decision making. While the routing decision is being

made, the rest of the data payload streams into the local data memory of the Ingress Processor. After the routing decision is made, the packet is sent into the Rotating Crossbar, which is implemented over the static network of the Raw processor. This data transfer may take multiple phases on the crossbar and hence a packet may be fragmented as it travels across the Rotating Crossbar. After the Rotating Crossbar has been traversed, the Egress Processor buffers the packet in its internal data memory until all of the fragments are available. Then it streams the completed IP packet to its output port, which is connected to an output line card.

4.4 Buffer Management

Practical design considerations that hinder and shape this design include the fact that each tile's data cache only has one port. Thus accessing a tile's data cache requires tile processor cycles, since there is no built-in Direct Memory Access engine from the networks into the data cache. For example, buffering data on a tile's local memory requires two processor cycles per word. Also, code running throughout this design is carefully unrolled, because even though there is no branch penalty for predicted branches, a branch still uses one cycle to execute on the tile processor.

This design is rather conservative with regards to computational resources, and it leaves room to grow and hence possibilities of using this same basic design for a core router. One of the challenges of this design is the aggravation of problem of packet queueing when doing core routing. This design assumes that there is large amount of buffering on the input and output external to the Raw Processor. This needs to be done because the maximum internal storage of the Raw Processor prototype is 2 megabytes. While this is a large amount for a single processor, the bandwidth-delay product for multigigabit flows is two to three orders of magnitude larger. Therefore in this design prototype, first-in-first-out delivery is implemented, with dropping assumed to be occurring externally to the Raw chip.

Chapter 5

Switch Fabric Design

This chapter moves from general descriptions to specifics, describing the design of the router's switch fabric and the Rotating Crossbar algorithm. Several sections show the properties of this algorithm, including fairness and absence of possible deadlocks.

5.1 Rotating Crossbar Algorithm

A part of the problem was to design an algorithm that would allow the use of the fast static networks to do dynamic routing.

It has been shown that Raw was suitable for streaming and ILP applications with patterns defined at compile time [11, 8], but the approaches to building dynamic applications were still to be researched. Several techniques were created and analyzed [2, 22], but unfortunately most of them either led to underutilization of the Raw processor, an unbalanced load distribution across the tiles, or to complicated configuration analysis in order to determine and avoid possible deadlocks of the static networks.

The following is an explanation of the Rotating Crossbar algorithm with global knowledge, which is similar to a well-known Token Ring algorithm [7] that has been widely used in networking. In this case, however, it is nicely applied to the domain of router microarchitecture. The Rotating Crossbar algorithm allows to arbitrarily connect four Ingress Processors to four Egress Processors, provided there are no con-

flicts for Egress Processors and Rotating Crossbar static networks, for the duration of one quantum of routing time, which is measured by the number of 32-bit words to be routed around the Rotating Crossbar. Fortunately, this algorithm avoids the aforementioned undesirable features and is very efficient.

The algorithm is based on the idea of a token, which denotes the ultimate right of a Crossbar Processor to connect its respective Ingress Processor to any of the four Egress Processors of the Raw chip. The token starts out on one of the Crossbar Processors, called the master tile. However, there are no slave tiles, since, if the master tile is not sending its data, which can happen is case its incoming queue is empty, every downstream tile has an opportunity to fill in the existing slots in the static network, though the probability to send data is decreasing with every step down the stream. By using a token, we can avoid starvation of Ingress Processors, since it guarantees that each input will send at least once every four routing cycles. It is also important to notice here that the token does not actually get passed around the crossbar tiles. Instead, it is implemented as a synchronous counter local to each of the Crossbar Processors.

5.2 Rotating Crossbar Illustrated

In the beginning of each routing phase all four Crossbar Processors read their respective packet headers, which contain output port numbers prepared by the Ingress Processors after route lookup. In the next phase the Crossbar Processors exchange these headers with each other. In the following phase they stream their local data into the Rotating Crossbar depending on current tile's privileges, which are determined by a local copy of a global routing rule for a given combination of the master tile and four packet headers. We pipeline the process by overlapping the processing of the current header with the streaming of the previous packet's body into the crossbar. After the routing of the current time quantum is over, the token is passed to the next downstream crossbar tile, and the sequence repeats.

Figure 5-1 illustrates the idea of the Rotating Crossbar algorithm. Imagine that

Ingress Processors of Ports 0, 1, 2, and 3 have packets destined to Ports 2, 3, 0, and 1 respectively in their caches, and Port 0 has a token – shown is gray (see the top of Figure 5-1). There is a full-duplex tile-to-tile connection on Raw static network 1 between neighboring Crossbar Processors, which allows to simulteneously route data from all Ingress Processors to all Egress Processors, as shown in the botton of Figure 5-1. Here, the light gray illustrates the clockwise transfer of data, and the dark gray demonstrates the counterclockwise one. Port 1 needs to route data in the counterclockwise direction because Port 0, which is situated upstream, has already used the clockwise connection between Crossbar Processor 1 and Crossbar Processor 2. The same situation happens with Port 3 – it needs to use counterclockwise rotation to transfer its data as well.

5.3 Sufficiency of a Single Raw Static Network

It is important to notice that Figure 5-1 shows a best-case scenario, when all ports are able to send. However, an interesting topological property of the system is that whenever there is no contention for output ports, a single full-duplex connection between Crossbar Processors is sufficient to provide enough of interconnect bandwidth, and the use of the Raw second static network does not improve the performance of the router.

5.4 Fairness

An obvious and immediate advantage of this algorithm is its natural fairness, which eliminates the danger of starvation observed in other non-token-based algorithms. When there is no global control over the transmission of packets, upstream crossbar tiles can flood the static network and prevent downstream tiles from sending data. Furthermore, there are advantageous side effects of this approach. One of them is the ease of augmenting the functionality of the IP router with such important features as Quality of Service, flow prioritization and traffic shaping. These additions can

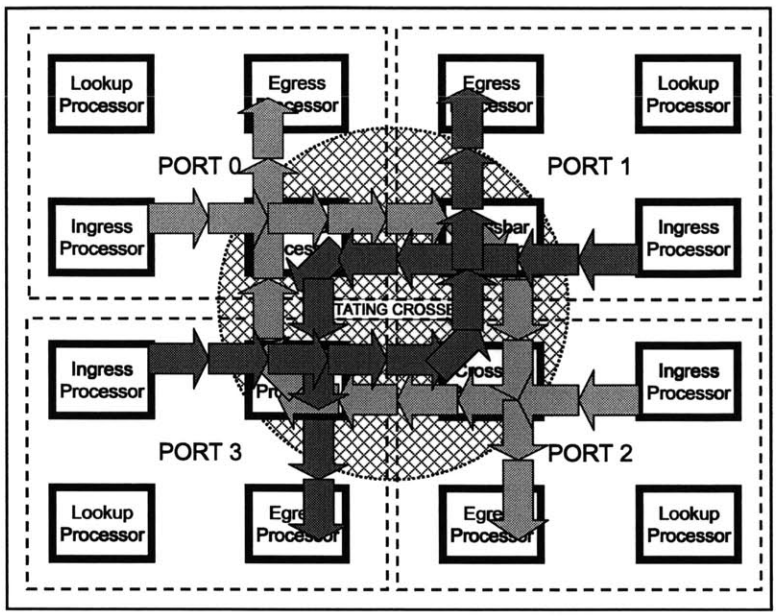


Figure 5-1: Rotating Crossbar illustrated. In this configuration all of the four Ingress Processors are sending data to Egress Processors.

be achieved by using a weighted round robin modification of the Rotating Crossbar algorithm. This can be done simply by allowing different ports a weighted amount of differing time with the token.

5.5 Deadlock Avoidance

While starvation can be overcome by using more complex macro-patterns proposed in other algorithms, another far more dangerous problem of deadlocking the static network is solved with this algorithm. The deadlock can occur when the data-flow between the Crossbar Processors forms a loop, and the static networks are not scheduled properly. However, the described algorithm can not deadlock the static network, because it only allows non-blocking crossbar schedules carefully generated at compile time (see further on for more information).

Chapter 6

A Distributed Scheduling Algorithm for the Rotating Crossbar

This chapter introduces a distributed scheduling algorithm for the Rotating Crossbar, and explains how the constraints on the memory system of the Raw processor influence on the implementation, and show a minimization of the configuration space made in order to fit the code in a tile's local instruction memory. This chapter also describes the timing of the algorithm at run-time, as well as the programming techniques used on the Crossbar Processors.

6.1 Defining Configuration Space

In the current router layout there are four input ports sending to four output ports, as shown in Figure 4-1. Therefore, assuming that the input queue can also be empty, and letting the number of possible token positions be equal to the number of crossbar tiles, the configuration space can be defined as

$SPACE = |Hdr_0| \times ... \times |Hdr_3| \times |Token|,$

where $|Hdr_0| = ... = |Hdr_3| = 5,$

and $|Token| = 4,$

which gives us $SPACE = 5^4 \times 4 = 2,500$

Thus, the necessary number of individual Crossbar Processor configurations is equal to 2,500. However, each tile of the Raw processor has only 8,192 words of local instruction memory and 8,192 words of switch memory, and storing the Crossbar Processor code outside of the chip is too slow for a gigabit router. Therefore, there are approximately 3.3 instructions left per each configuration, which is obviously not enough. Hence there needs to be an optimization applied to the configuration space, which would allow us to implement the router.

6.2 Minimizing Configuration Space

As an optimization of the configuration space we propose the definition given in Table 6.1. Rather than defining the space through possible combinations of packet headers and token owners, we change the focus to enumerating **clients**, or potential **incoming** occupants, of a Crossbar Processor's **servers** – static networks connecting a Crossbar Processor to its **outgoing** neighboring tiles, as shown in Figure 6-1.

The meaning of server names is the following: "out" is connection from a Crossbar Processor to an Egress Processor, "cwnext" and "ccwnext" are the clockwise and counterclockwise downstream networks around the crossbar respectively. Correspondingly, the meaning of the client names is: "in" is the network connecting an Ingress Processor with a Crossbar Processor, "cwprev" and "ccwprev" are the incoming networks to a Crossbar Processor from clockwise and counterclockwise neighbors.

Fortunately, not all possible configurations are used by the compile-time scheduler, which allows to decrease the number of distinct configurations even more. The aforementioned minimization cuts down the number of configurations by 78 times and creates a self-sufficient subset of 32 entries. Here, "out", "cwnext" and "ccwnext" have the same meaning, as in the previous paragraph. There also is a specific expansion number of a particular combination of clients which is necessary to keep track of relative distances of data sources to a Crossbar Processor (the assembly code of switch processors of the crossbar needs to be carefuly software-pipelined or loop-unrolled in

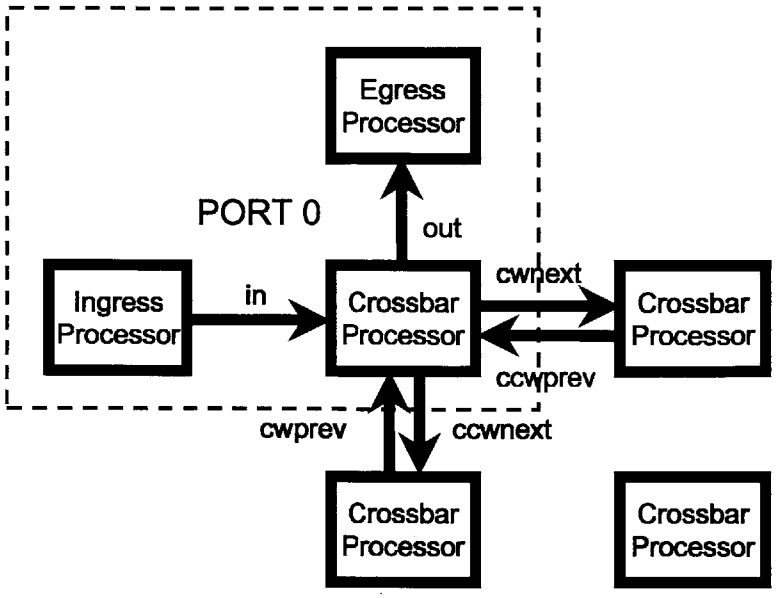


Figure 6-1: Network connections of a crossbar tile. Each Crossbar Processor has three incoming ("client") and three outgoing ("server") connections.

servers	`out, cwnext, ccwnext`
clients	$\emptyset$, `in, cwprev, ccwprev`

Table 6.1: Clients and servers of a Crossbar Processor.

order to avoid the deadlock of Raw static networks), as well as a special boolean value, which is set to TRUE in case an Ingress Processor can not send data in a given configuration.

6.3 Phases of the Algorithm

The sequence of events happening in Crossbar Processors is shown in Figure 6-2. They both are intended to facilitate the understanding of the assembly code examples given in the following section.

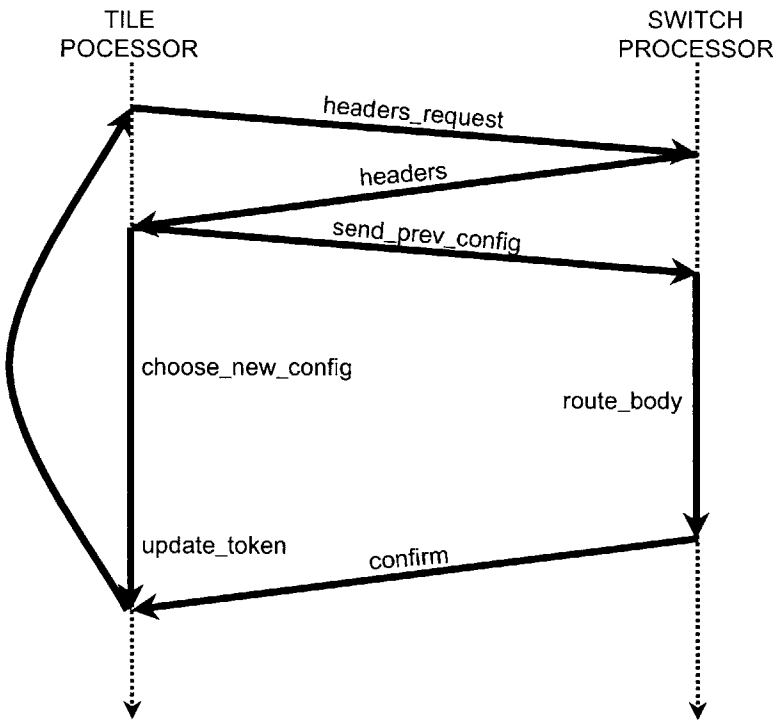


Figure 6-2: Phases of the Rotating Crossbar algorithm.

6.4 Designing an Automatic Compile-time Scheduler

In order to simplify code generation of the IP switch, we built a tool for automatic compile-time scheduling of crossbar configurations. The idea of this scheduler is a sequential walk starting from the master tile downstream across all crossbar tiles and filling in reservations for inter-crossbar and crossbar-to-output static network connections. When the reservations are fully filled with IDs of requesting crossbar tiles, there is another simplificaion pass implemented in accordance with the aforementioned space minimization. The resulting schedule is then converted to Raw assembly by the third pass.

6.5 Programming Tile Processors of the Rotating Crossbar

Each of the Raw tiles looks very much like a MIPS R4000, and the instruction sets of these two processors are also similar. The tile processor code is programmed with the use of software pipelining: the tile processor of the crossbar tile computes the address into the jump table of configurations while the switch processor is routing the body of the previous packet, then receives a confirmation from the switch processor stating that the routing is finished, reads the new set of headers and loads the address of the configuration into the program counter of the switch processor to immediately route the current body.

The second Raw static network, as well as the dynamic network, have not been used in the algorithm. As it was mentioned earlier, the addition of the second static network to the system does not improve the performance of the router because of the limiting factor of contention for output ports rather than insufficiency of inter-tile bandwidth.

Chapter 7

Results and Analysis

This chapter describes the results of our work – the peak and aggregate performance of the Raw Router compared to the Click router, which is another router implemented on a general-purpose processor. The chapter shows that we have achieved the goal of building a multigigabit router on Raw. This chapter also studies the efficiency of the current implementation and explains the utilization of the Raw processor on a per-tile basis. The analysis also suggests a general approach to obtain the maximum utilization of the router.

7.1 Gathering of Data

Due to the fact that the Raw processor is not physically available yet, we have implemented the router and tested it on the Raw simulator.

7.2 Peak Performance

Figure 7-1 demonstrates the peak performance compared to the Click Router. The performance of the router built on Raw general-purpose processor is two orders of magnitude better than the results obtained on Intel general-purpose processors making Raw general-purpose processor a viable candidate for networking applications.

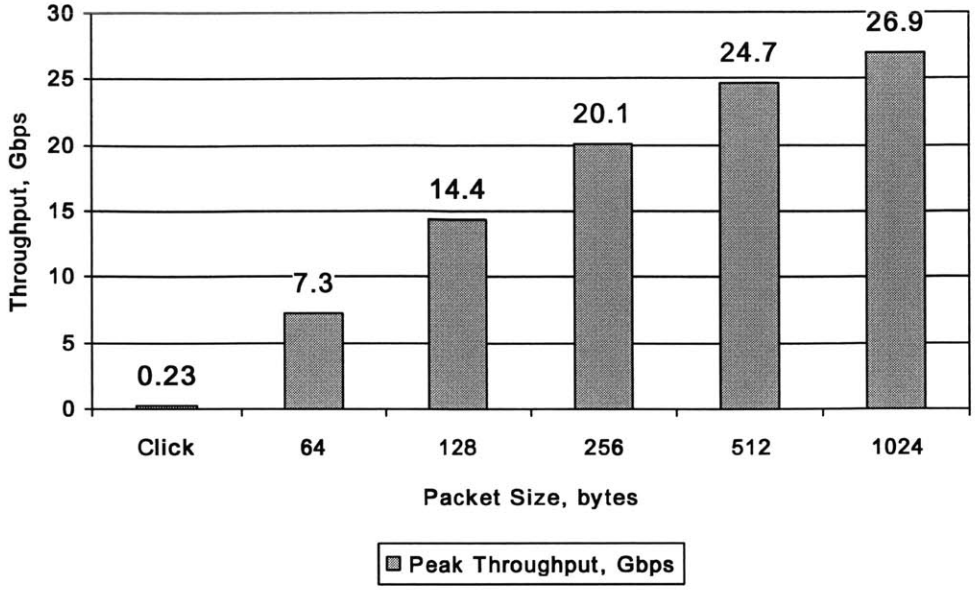
7.3 Average Performance

Figure 7-1 also shows the average performance compared to the Click Router. Note that the average performance is only about 69% of the peak performance due to the contention for output ports. It is also important to notice that these results are observed under complete fairness of the traffic.

7.4 Efficiency Study

There are several factors which contribute to the growth of performance when using larger packet sizes, but the most important one of them is certainly the relative amount of time that the static network is kept busy. In order to achieve better performance of the algorithm it is needed to decrease the processing overhead by spending less relative time in the tile processor and more on streaming data through the Raw processor networks. To see that this is true, let us take a look at Figure 7-3, which shows the utilization of the Raw processor when routing 64- and 1024-byte packets (the mapping of functional elements of the router to Raw tile numbers is given in Figure 7-2).

When routing 64-byte (the top of Figure 7-3) and 1024-byte (the bottom of the figure) packets, gray on tiles 4, 7, 8, and 11 means that the input ports are blocked by the crossbar. The top graph shows that Raw utilization is considerably lower for smaller packet sizes than for bigger packet sizes. It is possible to get close to Raw static network bandwidth limit when routing larger packets.



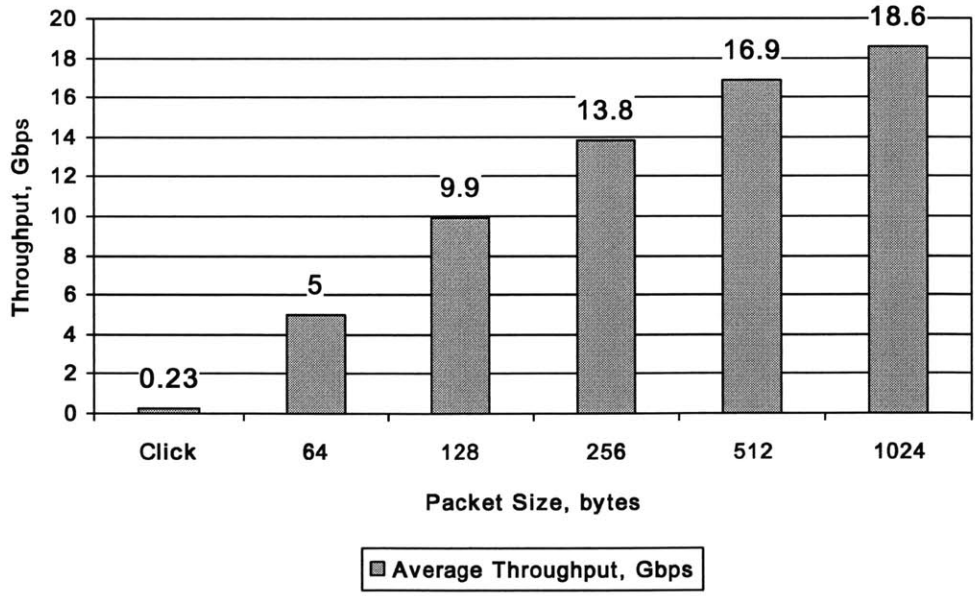


Figure 7-1: Router performance compared to the Click Router.

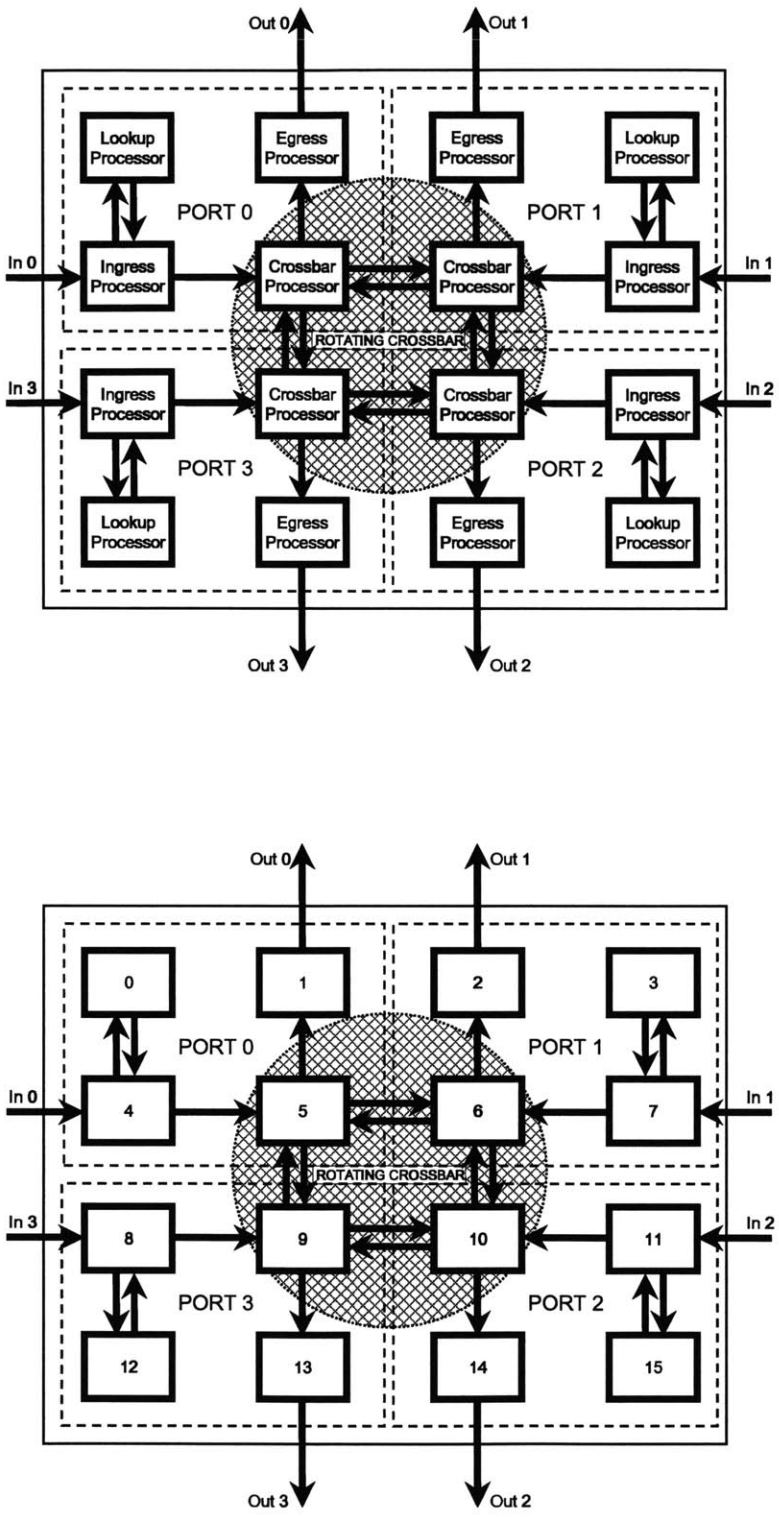


Figure 7-2: Mapping router functional elements to Raw tile numbers.

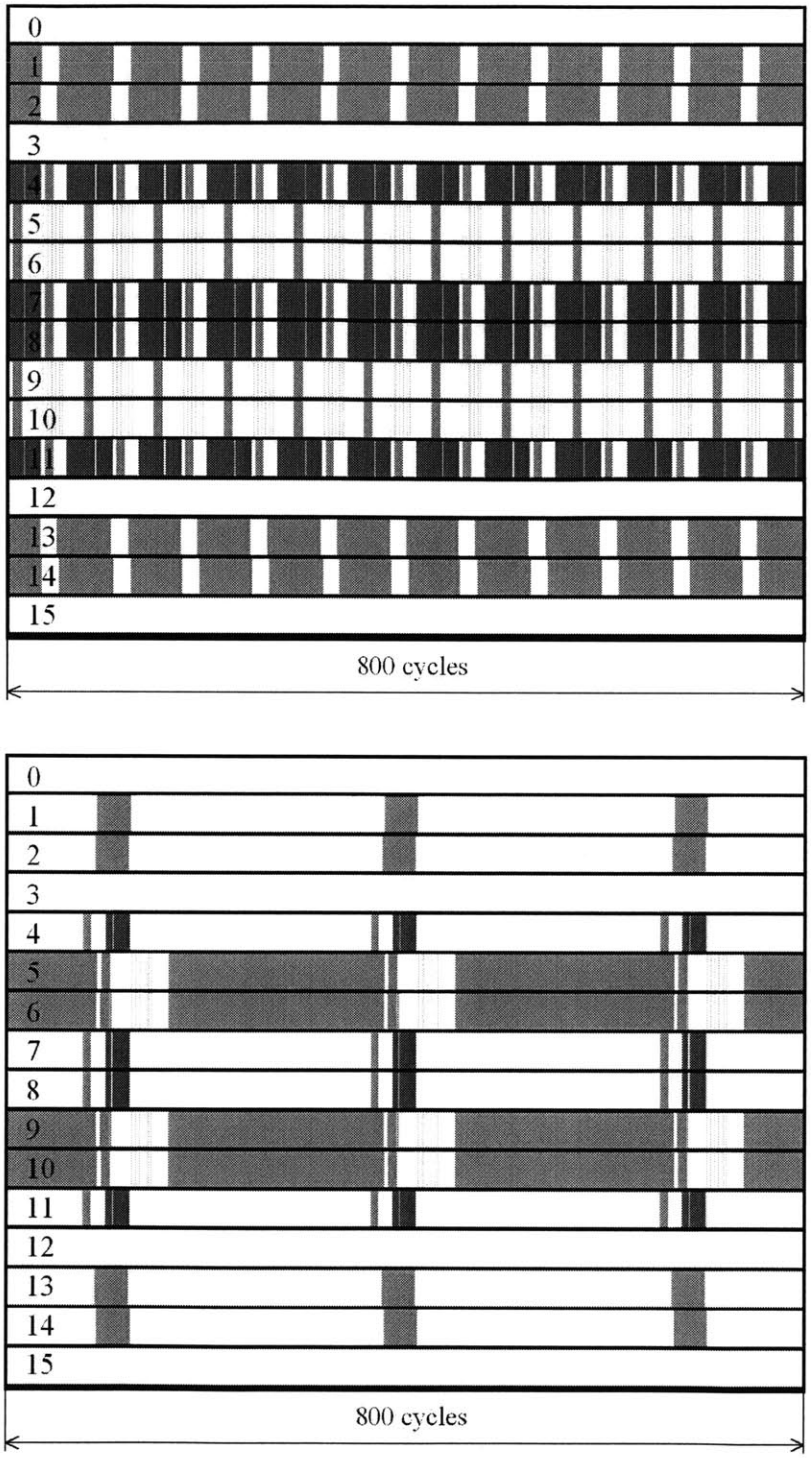


Figure 7-3: Utilization of the Raw processor on a per-tile basis. The top graph is for 64-byte packets, and the bottom graph is for 1,024-byte packets, both plotted for 800 clock cycles. The numbered horizontal lines correspond to Raw tile processors. Gray color means that a tile processor is blocked on transmit, receive, or cache miss.

Chapter 8

Future Work

This chapter decribes the future improvements that we are planning to add to the existing router, including new designs pursuing full utilization of the Raw processor, the implementation of the IP route lookup on Raw, the issues of scalability and support of multicast traffic in the switch fabric, flow prioritization to deploy Quality of Service, as well as the application of the current router layout for routing in low earch orbit satellite systems.

8.1 Pursuing Full Utilization of Raw

The results obtained so far are quite promising, but the Raw processor can do better, and there are resources to attain this goal. First of all, there is another static network waiting to be used for routing. Secondly, while the two static networks will be busy streaming data from inputs to outputs, the dynamic network of Raw can provide much help for control messaging and reconfiguration to reach the optimal performance.

8.2 Implementing IP Route Lookup

The previous sections described the solution to the problem of switching, but there still remains an issue of route lookup. We would like to look at various lookup algorithms with the hope of being able to support enough routes to compete as a core

router, such as the one given in [6]. To be able to do this, one or several tiles per input port will act as the route resolving entities. While network processors deisgned to do route resolution are multi-threaded, the Raw architecture is not multi-threaded, but its exposed memory system allows for the same advantages as a multi-threaded architecture. This main advantage is the ability to get work done while the processor is blocked on external memory accesses. On the Raw Processor, memory is simply implemented in a message passing style over one of the dynamic networks. Typically when accessing RAM with loads and stores, the cache is backed in a write-back manner by main memory, which is accessed by a small state machine that generates and receives messages on the memory dynamic network. If the programmer wants to use the system in a non-blocking nature, dynamic messages can be created and sent to the memory system without using the cache. Thus this provides the same advantage of non-blocking reads that a multi-threaded network processor provides.

8.3 Adding Computation on Data

Another future implementation feature is the building of an infrastructure to provide the ability to implement streaming based computations, such as encryption, inside of the switch fabric as it was described earlier. This will take the form of special bits in the headers that are exchanged around the routing ring. These bits describe to the switch fabric what form of computation needs to be applied.

8.4 Building Intelligent Routers

One more research direction that holds promise is the application of the computational power of Raw to more intelligent routers, such as providing endpoint network users more control over their communications. [25]

8.5 Scalability

The work presented here describes an architecture for a 4-input 4-output port router. While this is a good starting point, one goal of this research is to also examine larger configurations. The Raw architecture itself was designed to be a scalable computational fabric, and this is the route that will be needed to be followed to build a scalable router. Building this larger fabric of processors is as simple as gluelessly connecting multiple Raw chips in a two dimensional mesh grid. One solution is simply to build a larger router out of multiple of these small 4-port routers, or at least out of multiple 4-port crossbars.

8.6 Supporting Multicast Traffic

It is becoming increasingly important for a router to support multicast traffic, and we are planning to add this functionality to the existing Rotating Crossbar algorithm by allowing a single Ingress Processor to send data to several Egress Processors simultaneously. This modification is trivial considering the ease of programmability of the switch fabric.

8.7 Quality of Service

As mentioned earlier, we are also going to implement the prioritization of packet flows from Ingress Processors. This is easily done by letting Ingress Processors include priority information into the local header sent to Crossbar Processors, and adding the arbitration code into the code running on the switch fabric.

8.8 Routing in Low Earth Orbit Satellite Networks

Several strategies have been proposed for routing in a low earth orbit (LEO) satellite system [9, 16, 23, 24]. Some of them are based on the Internet Protocol (IP) and

asynchronous transfer mode (ATM) switching. However, issues like memory requirements of the satellites in the LEO network and the overheads involved in transmitting packets over the network have frequently been ignored. We are planning to look at developing an efficient solution to the routing issue in a LEO network using general-purpose processors like Raw.

Chapter 9

Conclusion

The presented work shows that efficient routing can be done on the programmable static network of the Raw general-purpose processor. The results obtained in the simulation demonstrate that a 4-port edge router running on a 250 MHz Raw processor is able to switch 3.3 million packets per second at peak rate, which results in the throughput of 26.9 gigabits per second for 1,024-byte packets, suggesting that it is possible to use the Raw Processor as both a network processor and switch fabric for multigigabit routing. Mixing computation and communication in a switch fabric lends itself to augmenting the functionality of the router with encryption, compression, intrusion detection, multicast routing, and other valuable features. The presented Rotating Crossbar algorithm displays good properties, such as fairness and scalability, and allows for further improvement by taking advantage of the second static network of the Raw general-purpose processor. It is also naturally capable of accommodating the implementation of Quality of Service. Therefore, we conclude that the Raw processor will be further explored in order to add more of these features.

Bibliography

[1] Matthew Adiletta. The Intel Network Processor (IXP): Yesterday and Today, April 2002. Presentation at the MIT LCS Computer Architecture Group.

[2] Gleb A. Chuvpilo, David Wentzlaff, and Saman Amarasinghe. Gigabit IP Routing on Raw. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture, Workshop on Network Processors*, February 2002.

[3] The Evolution of High-End Router Architectures: Basic Scalability and Performance Considerations for Evaluating Large-Scale Router Designs. *White Paper, Cisco Systems*, January 2001.

[4] Cisco 12000 Gigabit Switch Router. *White Paper, Cisco Systems*, 1997.

[5] William J. Dally. Wire-Efficient VLSI Multiprocessors Communication Networks. In *Proceedings of the Stanford Conference on Advanced Research in VLSI.* MIT Press, 1987.

[6] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small Forwarding Tables for Fast Routing Lookups. In *ACM SIGCOMM*, September 1997.

[7] Robert Donnan. *IEEE Standard 802.5-1989, IEEE Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specifications.* 1989.

[8] Michael Gordon, William Thies, Michal Karczmarek, Jeremy Wong, Henry Hoffmann, David Z. Maze, and Saman Amarasinghe. A Stream Compiler for

Communication-Exposed Architectures. In *Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[9] Yukio Hashimoto and Behcet Sarikaya. Design of IP-based Routing in a LEO Satellite Network. In *3rd International Workshop on Satellite-Based Information Services, Mobicom'98*, October 1998.

[10] Eddie Kohler. *The Click modular router.* PhD thesis, MIT, Cambridge, MA, June 2000.

[11] Walter Lee, Rajeev Barua, Matthew Frank, Devabhatuni Srikrishna, Jonathan Babb, Vivek Sarkar, and Saman Amarasinghe. Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine. In *Proceedings of the Eighth ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 46–57, San Jose, CA, October 1998.

[12] Nick McKeown. Fast Switched Backplane for a Gigabit Switched Router. Technical report, Stanford University.

[13] Nick McKeown. *Scheduling Cells in an Input-Queued Switch.* PhD thesis, University of California at Berkeley, May 1995.

[14] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. In *Proceedings of the Symposium on Operating Systems Principles*, pages 217–231, 1999.

[15] Donald Morrison. PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the ACM*, October 1968.

[16] Paolo Narvaez, Antonio Clerget, and Walid Dabbous. Internet Routing over LEO Satellite Constellations. In *3rd International Workshop on Satellite-Based Information Services, Mobicom'98*, October 1998.

[17] C. Partridge, P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohlami, T. Ma, J. Mcallen, T. Mendez, W. Milliken, R. Osterlind, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G. Troxel, D. Waitzman, and S. Winterble. A Fifty Gigabit Per Second IP Router. In *IEEE/ACM Transactions on Networking*, 1997.

[18] Craig Partridge. *Gigabit Networking.* Addison Wesley Publishers, 1994.

[19] Y. Tamir and G. Frazier. High Performance Multiqueue Buffers for VLSI Communication Switches. In *Proceedings of the 15th Annual Symposium on Computer Architecture*, June 1998.

[20] Michael B. Taylor. Design Decisions in the Implementation of a Raw Architecture Workstation. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, September 1999.

[21] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal. Baring It All to Software: Raw Machines. *IEEE Computer*, 30(9):86–93, September 1997. Also available as MIT-LCS-TR-709.

[22] David Wentzlaff, Gleb A. Chuvpilo, Arvind Saraf, Saman Amarasinghe, and Anant Agarwal. RawNet: Network Processing on the Raw Processor. In *Research Abstracts of the MIT Laboratory for Computer Science*, March 2002.

[23] Markus Werner. A Dynamic Routing Concept for ATM-Based Satellite Personal Communication Networks. In *IEEE Journal on Selected Areas in Communications*, October 1997.

[24] Markus Werner, Cecilia Delucchi, HansJorg Vogel, Gerard Maral, and Jean-Jacques De Ridder. ATM-Based Routing in LEO/MEO Satellite Networks with Intersattelite Links. In *IEEE Journal on Selected Areas in Communications*, January 1997.

[25] David Wetherall, Ulana Legedza, and John Guttag. Introducing New Internet Services: Why and How. *IEEE Network*, July 1998.