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

Gleb A. Chuvpilo and Saman Amarasinghe
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
{chuvpilo, saman}@lcs.mit.edu

Abstract

The switching of packets and other performance-critical tasks in modern Internet routers are done done using Application Specific Integrated Circuits (ASICs) or custom-designed hardware, 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 paper proposes the use of the Raw general-purpose processor as both a network processor and switch fabric for multi-gigabit routing. We show that 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 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, 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 distinct 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 paper proposes the use of the Raw general-purpose processor [11] as both a network processor and switch fabric to build a Raw Router [1]. The Raw processor, through its tiled architecture and software-exposed on-chip networking, has enough internal and external bandwidth to deal with multigigabit routing.

2 Raw Processor

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

2.1 Processor Layout

The Raw processor is an array of 16 identical, programmable tiles (Figure 1). A tile (Figure 2) contains an 8-stage in-order single-issue MIPS-style compute processor, a 4-stage pipelined FPU, a 32kB data cache, two types of communication routers — static and dynamic, and 96kB of instruction cache. These tiles are interconnected to neighboring tiles using four full duplex 32b networks, two static and two dynamic. The static router controls the static networks, which are used as point-to-point scalar transport for data between the tiles. The dynamic routers and networks

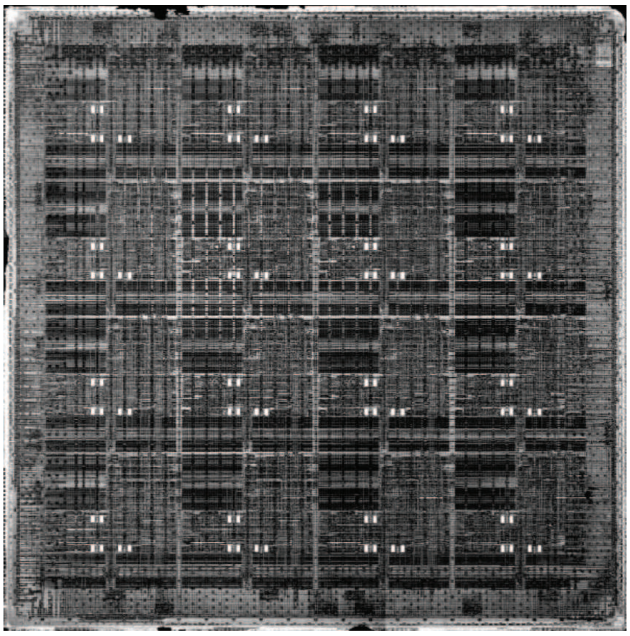

Figure 1. Raw processor micrograph (256mm^2).

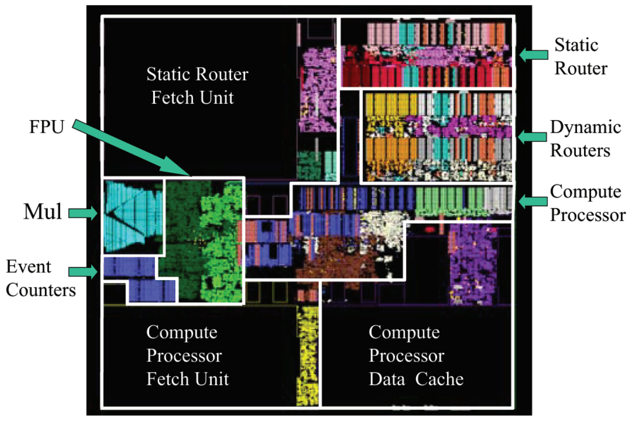

Figure 2. Raw tile layout.

are used for all other traffic such as memory, interrupts, I/O, and message passing codes.

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 instruction 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. Larger Raw systems can be designed by stamping out more tiles.

The Raw chip 16-tile prototype is built in IBMs SA-27E, a 0.15μm, 1.8 V, 6-level Cu ASIC process. Raw has a 1657 pin CCGA package. These HSTL pins provide 14 full-duplex 32-bit chipspeed channels that can be connected to either DRAM or stream I/O devices. The Raw chip's core has been verified to run programs at 420 MHz. More information on the Raw microarchitecture can be found in the Raw Processor Specification. [10]

2.2 Communication Mechanisms

The main communication mechanism in the Raw Processor is the static network. The code sequence to send a value over a static network takes five cycles to execute. During the first cycle, the source tile sends a value to its static router. During the second cycle, this router transmits the value to the adjacent router. During the third cycle, the latter transmits the value to its compute processor. During the fourth cycle, the value enters the decode stage of the processor. During the fifth cycle, the value can be consumed. 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 static router 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 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.

The dynamic networks on Raw are used for communication 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, 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 Raw Router Architecture

This and the following sections examine the Raw Router architecture and a complete router configuration. This section 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.

3.1 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.

3.2 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 onto the Raw tiles. This mapping should balance the computation load between the 16 compute processors of Raw. More importantly, the mapping has to efficiently support the communication patterns of the router.

Figure 3 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

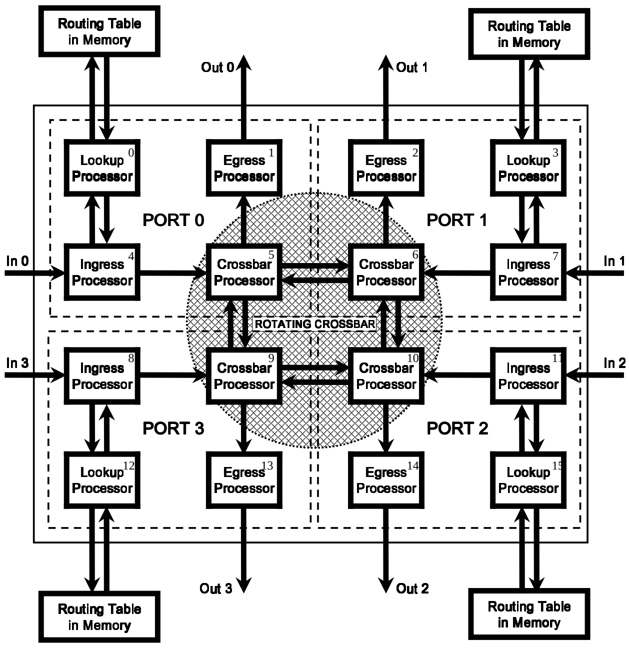

Figure 3. Mapping router functional elements to Raw tiles.

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, service 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$\times$4 IP router.

3.3 Data Path

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 for 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.

3.4 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 compute 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, the assembly code should be carefully unrolled, because even though there is no branch penalty for predicted branches, a branch still uses one cycle to execute on the compute processor.

This design is rather conservative with respect 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.

4 Switch Fabric Design

This section 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.

4.1 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 [8, 6], but the approaches to building dynamic applications were still to be researched. Several techniques were created and analyzed [2, 12], 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 [5] 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 conflicts 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 in 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.

4.2 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.

4.3 Sufficiency of a Single Raw Static Network

An interesting topological property of the router 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.

4.4 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 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.

4.5 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).

5 A Distributed Scheduling Algorithm for the Rotating Crossbar

This section 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 section also describes the timing of the algorithm at run-time, as well as the programming techniques used on the Crossbar Processors.

5.1 Defining Configuration Space

In the current router layout there are four input ports sending to four output ports, as shown in Figure 3. 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.

5.2 Minimizing Configuration Space

As an optimization of the configuration space we propose following definition: 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 4.

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 order to avoid

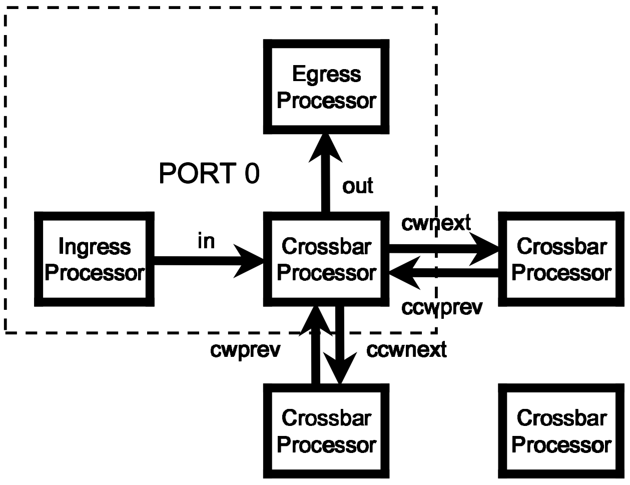

Figure 4. Network connections of a crossbar tile. Each Crossbar Processor has three incoming ("client") and three outgoing ("server") connections.

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.

5.3 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.

5.4 Programming the Tiles 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 compute processor code is programmed with the use of software pipelining: the compute 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.

6 Results and Analysis

This section describes the results of our work – the peak and aggregate performance of the Raw Router compared to the Click router [9, 7], which is a sofware router implemented on a general-purpose processor. The section shows that we have achieved the goal of building a multigigabit router on Raw. This section 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.

6.1 Peak Performance

Figure 5 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.

6.2 Average Performance

Figure 6 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.

6.3 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 compute 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, which shows the utilization of the Raw processor when routing 64- and 1024-byte packets.

When routing 64-byte (the top of Figure 7) and 1024-byte (the bottom of the figure) packets, gray on tiles 4, 7, 8,

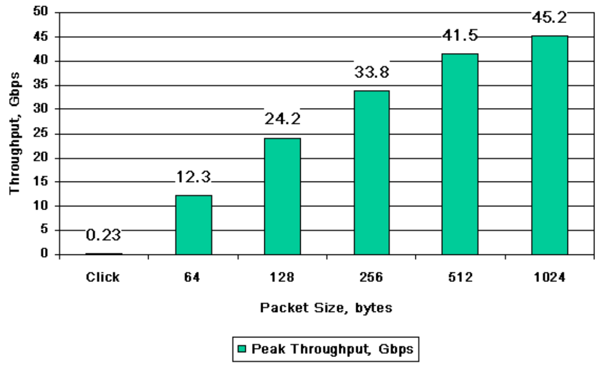

Figure 5. Peak Raw Router performance.

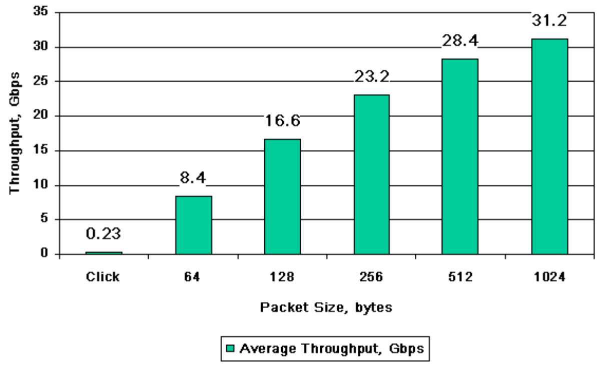

Figure 6. Average Raw Router performance.

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.

7 Future Work

This section 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 earth orbit satellite systems.

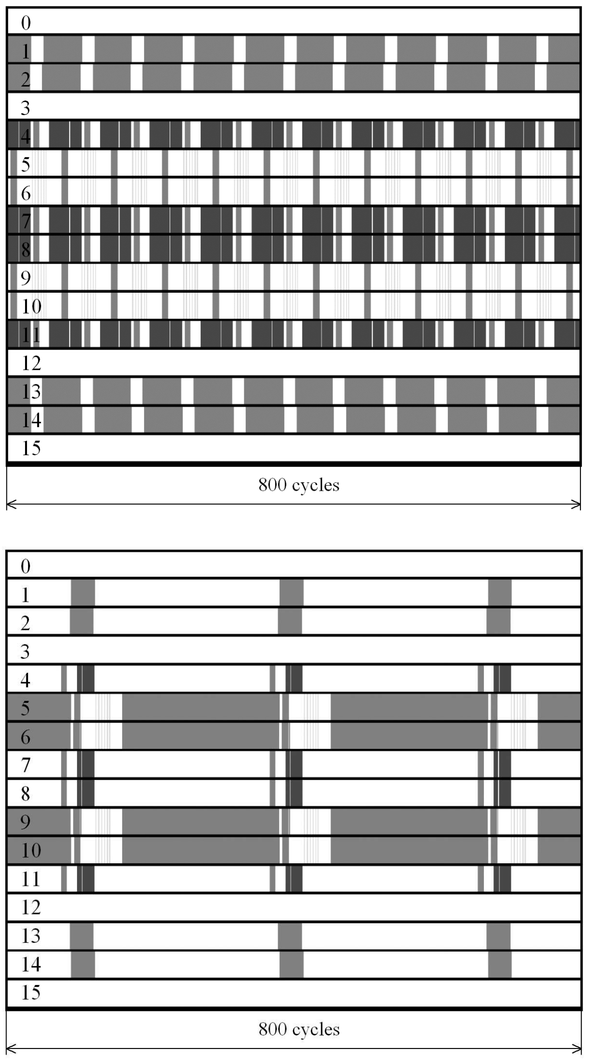

Figure 7. Utilization of the Raw processor on a per-tile basis. The top graph is for 64-byte packets, and the bottom graph is for 1,024-byte packets, both plotted for 800 clock cycles. The numbered horizontal lines correspond to Raw compute processors. Gray color means that a compute processor is blocked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

7.1 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 [4]. 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.

7.2 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 Conclusion

The presented work shows that efficient routing can be done on the programmable static network of the Raw general-purpose processor. The results demonstrate that a 4-port edge router running on a 420 MHz Raw processor is able to switch 5.5 million packets per second at peak rate, which results in the throughput of 45.2 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.

References

[1] G. A. Chuvpilo. High-Bandwidth Packet Switching on the Raw General-Purpose Architecture. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, August 2002.

[2] G. A. Chuvpilo, D. Wentzlaff, and S. 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] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small Forwarding Tables for Fast Routing Lookups. In *ACM SIGCOMM*, September 1997.

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

[6] M. Gordon, W. Thies, M. Karczmarek, J. Wong, H. Hoffmann, D. Z. Maze, and S. Amarasinghe. A Stream Compiler for Communication-Exposed Architectures. In *Proceedings of the ACM ASPLOS*, 2002.

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

[8] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. 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, Oct. 1998.

[9] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. In *Proceedings of the Symposium on Operating Systems Principles*, pages 217–231, 1999.

[10] M. 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.

[11] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring It All to Software: Raw Machines. *IEEE Computer*, 30(9):86–93, Sept. 1997. Also available as MIT-LCS-TR-709.

[12] D. Wentzlaff, G. A. Chuvpilo, A. Saraf, S. Amarasinghe, and A. Agarwal. RawNet: Network Processing on the Raw Processor. In *Research Abstracts of the MIT Laboratory for Computer Science*, March 2002.