Gigabit IP Routing on Raw

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

Abstract

Network processors afford a great degree of flexibility to current day routers, yet they still have followed the trend of being largely specialized to the domain of route table look-up. By using a processor architecture that is more general purpose, routers can gain from economies of scale and increased programmatic flexibility. We propose the use of the Raw Processor [5] as both a network processor and switch fabric for multi-gigabit IP routing. While previous general purpose architectures have failed to give a useful programmatic interface to sufficient bandwidth to support multi-gigabit IP routing, the Raw Processor, through its tiled architecture and software exposed on-chip networking, has enough internal and external bandwidth to deal with high rate routing problems. We describe why the integration of processing elements and switch fabric leads to increased flexibility for one chip router solutions, and propose a rotating crossbar design that achieves efficient IP routing on the Raw static network.

1. Introduction

This paper presents the beginning of research in mapping real world applications normally only handled by custom hardware or special purpose processors to the Raw general purpose processor. The application area that this paper focuses on is mapping an IP router to Raw. The IP router discussed in this paper is designed to be either an edge router or just the switch fabric of a core router.

This paper has two main goals. Firstly, it shows that one can efficiently map a semi-dynamic communications pattern, such as the switch fabric of an IP router, to a programmatically static interconnect. And, secondly it proposes the incorporation of computation into the communication interconnect of an IP router's switch fabric. The addition of computation is motivated by two reasons. First, there is a growing need for routers to operate on the data payload of IP packets to provide extended services such as encryption. Second, 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.

2. The Raw Processor Overview

Before this paper goes into great detail about mapping IP routing onto the Raw Processor, the resources provided and organization of the Raw Processor need to be discussed. The Raw Processor is a general purpose microprocessor being developed in the Computer Architecture Group at The Massachusetts Institute of Technology.

The general organization of the Raw Processor is as a chip multiprocessor with multiple fine grain, first class, register mapped communication networks. The processor contains 16 tiles in a 4x4 mesh grid. A tile consists of a main 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 (ISA) 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 other ISAs' approaches 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 gluelessly connected to build fabrics of up to 1024 tiles.

The Raw project has a very software centric approach as can be seen in two of the main goals of the Raw chip.

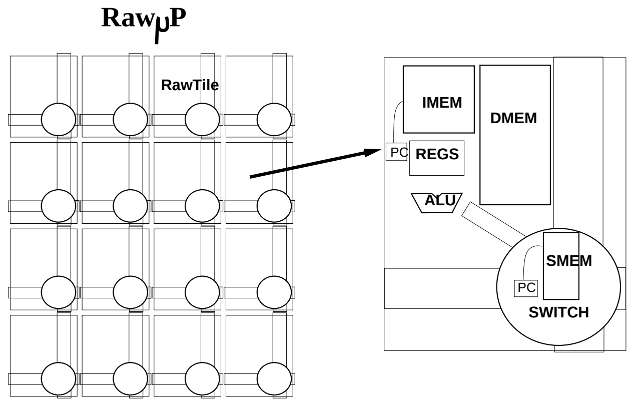

Figure 1. The Raw Processor is composed of 16 identical Tiles connected by two dynamic and two static communication networks.

The first goal is to reduce the complexity of the hardware as much as possible while maintaining high performance and do as much as can done in software in software. This goal allows for easier design and verification of hardware. It also allows for greater configurability. When algorithms are put in silicon, they are no longer changeable and as such lose opportunities to optimize based on the running application. An example of this software centric hardware approach is Raw's software based instruction caching system. The second goal that fits into this software centric hardware is the goal of exposing as much hardware resources up to the software level, and programmer's control, as is possible. The only substantial non-exposed state of the Raw Processor is the pipeline registers. Also, the on-chip networks extend off-chip, thus providing a direct software interface to the pins on Raw. This organization removes the inefficiencies present in other architectures that require the use of memory hierarchies to communicate with off chip devices.

Each Raw chip has 16 main processors, one in each tile. A main processor is a 32bit 8-stage pipelined MIPS-like processor. Each main processor contains a fully pipelined two-stage integer multiplier, and a pipelined four-stage single precision floating point unit. The main 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 main 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 main 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 main communication mechanism in the Raw Processor is the static switch network. This network is controlled by a simplistic 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 simplistic branches. Thus it is very well suited for compile time known communication patterns and is able to handle these without the need for headers as are found in dynamic networks. Each tile contains one switch 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 main 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, 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, while the static network's latency is only three cycles.

As can be seen from the proliferation of networks, the Raw Processor is filled with communication mechanisms and it is these communication mechanisms that allow it to exploit parallelism differently from other chip multiprocessors. For instance instead of simply allowing thread level parallelism, Raw's fine grain communication networks enable instruction level parallelism (ILP) to be mapped across multiple tiles efficiently. This is done by having advanced parallelizing compilers statically map a program's data-flow graph onto the Raw Processor's computation and communication resources. This is in contrast to superscalars which dynamically do functional unit assignment. Another application area where Raw excels is Streaming applications. These applications are typically digital signal processing applications such as software radios. Because of the internal and external bandwidth, and the streaming nature of the Raw hardware, these applications can receive large amounts of speed-up from the Raw Processor. An effort has been started to compile streaming applications to Raw using advanced compiler technology in the StreaMIT project. [4]

Lastly, this section will conclude with a note about the performance of the Raw Processor. The Raw Processor Prototype is being fabricated on IBM's SA-27E, 6 layer metal copper 0.15μ process. The Raw Processor will operate at or above 225MHz, 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 I/O pins. More information on the Raw microarchitecture can be found in the Raw Processor Specification. [3]

3. Router Design Overview

The goal of this project is to design a multi-gigabit single chip router solution using the Raw Processor. Some assumptions and practical considerations have influenced the design of this router. Firstly, 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 route look-ups require more analysis. Another design point is that this design is for a 4-input and 4-output router. Lastly, because this design is not for a core router, the route look-up tree is assumed to fit within a single tile's data cache.

Whenever designing a system, a high-level organizational plan is needed to guide overall design. This is even more apparent in this project where a system is being mapped to a fixed set of resources. The fact that these resources not only have logical connection constraints, but also spatial constraints makes this mapping even more difficult. Because Raw is a parallel processor, it only makes sense to statically map computation onto its parallel resources and use the native on-chip communication primitives to shuttle data around. Figure 2 shows graphically the mapping that was chosen for this project. Each input port uses two tiles, one to stream in and buffer data and another one to perform route look-ups. Four tiles are used as a crossbar as is shown in the center of the chip, and each output has a dedicated tile to deal with fragmentation internal to the router and stream data to the output line-card.

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 input's neighboring route processor to do classification and route decision making. While the routing decision is being made, the rest of the data payload streams into the input tile's data cache. After the routing decision is made, the packet is sent into the rotating crossbar which is implemented over Raw's static network. This may take multiple macro-cycles on the crossbar and hence a packet may be

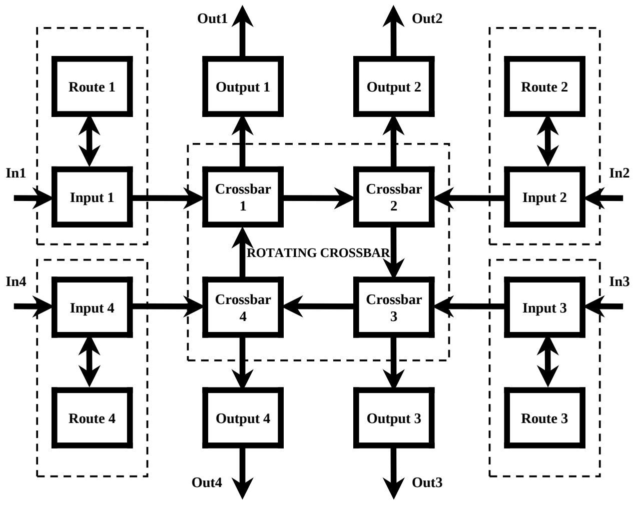

Figure 2. Layout of the router

fragmented as it travels across the crossbar. After the crossbar has been traversed the output tile buffers the packet in its internal cache 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.

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 main processor cycles because there is no built in DMA engine from the networks into the data cache. A side effect of this that may not be apparent at first glance is that when a tile is being used to buffer data as is being done in the input and output tiles of this design, two cycles are needed on the main processor for each data word, one to write and one to read data thus cutting the effective bandwidth in half. Also, code running throughout this design needs to be carefully unrolled, because even though there is no branch penalty for predicted branches, a branch still uses one cycle to execute on the main 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 challenge that faces this design as an edge router that is intensified when used as a core router is the problem of where to do bulk packet queuing. This design assumes that there is large amounts 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 2MB. While this is a large amount for a single processor, the bandwidth delay product for multi-gigabit flows is two to three orders of magnitude larger. Therefore in this design prototype, FIFO delivery is implemented, with dropping assumed to be occurring external to the Raw chip.

4. Routing Algorithm

The heart of the problem of designing an IP router on Raw was finding an algorithm that would allow the use of the fast static networks to do dynamic routing. Indeed, the fact that Raw was suitable for streaming applications with patterns defined at compile time was well known to the members of the Raw group, but the approaches to building dynamic applications were still to be researched. Several techniques were created and analyzed, but unfortunately most of them either led to 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 that arbitrarily connects four inputs to four outputs and avoids these undesirable features. It is similar to a well-known Token Ring algorithm [2] that has been widely used in networking, but in this case it is applied to the domain of router microarchitecture.

The algorithm is based on the idea of a token, which denotes the right of a crossbar tile to connect its respective input tile to any of the four output tiles of the processor. The token is implemented as a packet header field located in a single 32-bit word together with the destination port number. The token starts out on one of the crossbar tiles, called the master tile, while the other three crossbar tiles serve as passive transceivers, called slave tiles. The master crossbar tile first reads the header from its input tile. It then determines whether the packet should be routed immediately to its own output tile outside of the ring, or passed on to the next crossbar tile so that the packet would rotate clockwise around the geometrical center of the processor until it finds the desired output. All of the slave crossbar tiles on the way of the packet towards its destination perform the same check, but they do not read any more new packets from their respective input tiles. After the token has been used for one packet, it is passed on to the next crossbar tile in the ring, and the sequence repeats.

Figure 3 illustrates the idea of the algorithm. There are two packets coming into the crossbar from input 1 and input 2, which are destined for output 3 and output 2, respectively. The master crossbar tile is number 1 (gray color), and the direction of rotation is clockwise. According to the algorithm the crossbar tile having the token has the exclusive right to connect its input to any of the four outputs. Therefore, the packet coming from input 1 will be routed around the crossbar until it reaches crossbar tile 3, then it will be sent out of the ring and arrives at output 3. The token gives the right to read only one packet from the input and then it should be passed on to the next tile in the crossbar, which in this case is crossbar tile number 2. Finally, the packet coming from input 2 is routed to the desired destination tile, output 2.

An obvious and immediate advantage of this algorithm is its natural fairness, which eliminates the danger of starvation observed in other algorithms. When there is no control over the transmission of packets, upstream crossbar tiles can flood the static networks and prevent downstream tiles from sending data. While starvation can be overcome by using more complex macro-patterns, another far more dangerous problem of deadlocking the static network is solved with this algorithm. The deadlock occurs when the data-flow between the tiles of the crossbar forms a loop. Unfortunately, the negative side of this rotating crossbar algorithm is a certain underutilization of the available bandwidth of the processor.

In the current implementation, only one of the two static networks of the Raw Processor is used. Work on a version of the algorithm which will take advantage of both static networks by introducing a second token coexisting with the first one is now in progress. An issue with this improvement is again deadlock, but now the problem can be straightforwardly tackled. In this improvement only one of the master crossbar tiles (tile A) will need to check if the other tile with a token is sending. If this is found to be the case, tile A will need to prioritize the incoming traffic and route it to the desired output tile. Only then can it continue to send its own data, and no deadlock conditions occur because the other crossbar tile has used its token and will not send any more packets that could form a loop.

This rotating crossbar algorithm has proven to be a general, scalable, and powerful solution to designing an IP router on the Raw Processor. There are still several open questions to be researched, such as the aforementioned improvements to the algorithm, but it is evident that the ideology of having localized decision making distributed across the independent tiles of the processor pays off in terms of fairness. Furthermore, there are advantageous side effects of this approach including the ease of augmenting the functionality of the IP router with such needed features as Quality of Service 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. Flexibility of a General Purpose Switching Fabric

A serious disadvantage of current day network processors is that while they excel at doing route table look-ups and classifications quickly, they are not very adept with more computationally intensive calculations or computations that operate on the data payloads of packets. When these computations are needed in many current routers, they

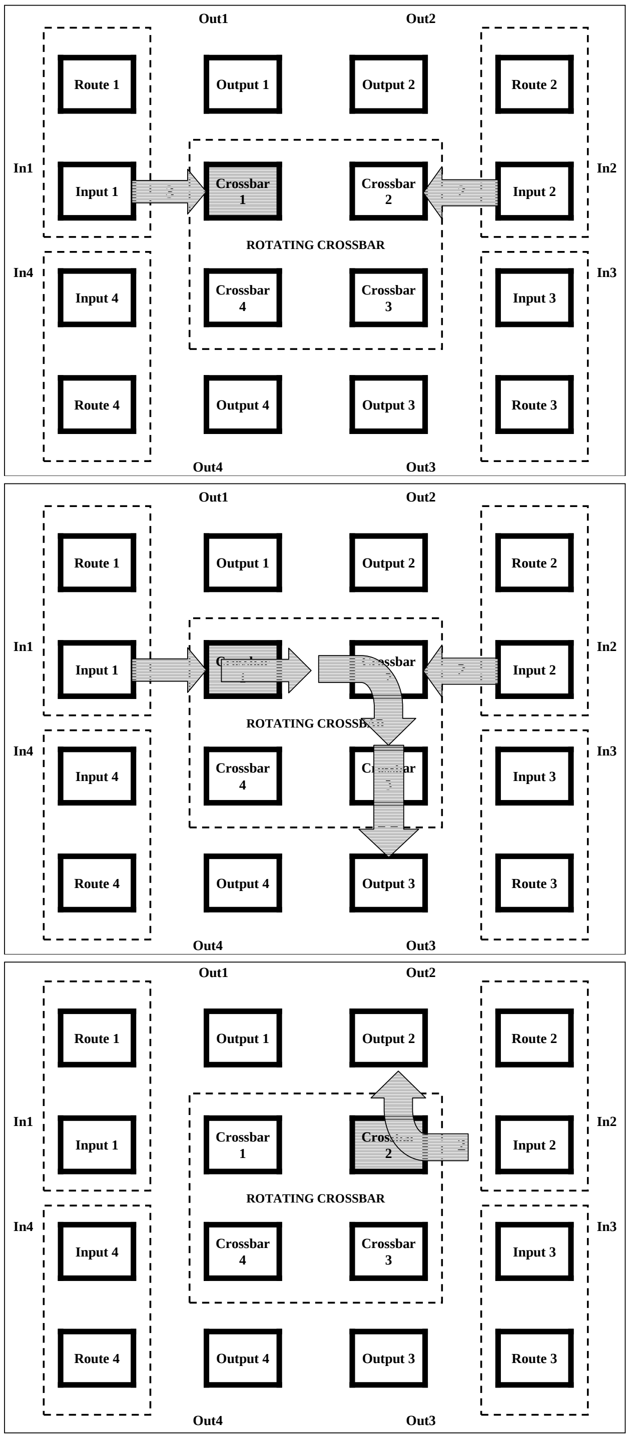

Figure 3. Rotating Crossbar illustrated

fall on a general purpose processor quaintly known as the ”slow path.” Because Raw is a general purpose processor, it is excellent at computing on the data while it in transit.

The property that Raw has copious amounts of computation and that that computation is inter-mixed with the switch fabric enables the switch fabric to compute on packets. This provides a nice property that the data needs to pass these computational resources while traversing the switch fabric so might as well compute on it. This is in contrast to standard router architectures where a good portion of the effort is getting the data to be computed on to where the computational resources are.

One foreseeable use of this form of intra-switch fabric computation is in implementing virtual private networks (VPN). One of the primary goals of VPNs is to provide secure communication over an insecure medium like the Internet. To achieve this, encryption is needed on both sides of the virtualized links. These encryption algorithms need to operate on the data payload of an IP packet, which can be difficult on standard routers because of their less than copious amounts of computation. Raw’s balance of computation and communication makes it suitable for this sort of computation. This sort of payload computation is not completely specific to this paper’s proposed router, but rather may be an impetus to mix computation and communication on future switch fabrics.

Mixing computation and communication in switch fabrics is not only useful for encryption, but has some other interesting uses. One of these is intrusion detection systems (IDS). An IDS is typically used in conjunction with a firewall to detect intrusion attempts against a protected network. They typically require analysis of higher level protocols and long term state. With computation integrated inside of a router’s switch fabric, an IDS could be crafted to work by simply sniffing the traffic that it finds ”interesting” and over time keep track of connections that are possible intrusions.

Lastly, mixing computation inside of a routers switch fabric can be used for compression. An idle node in a switch fabric could decide to compress traffic thus saving bandwidth on certain links. This would require some protocol to support this intra-network compression, or it could be implemented inside of one administrative domain in a user transparent manner. Also, because of the internal state of the auxiliary computation in the switch fabric, there is possibility of per-flow compression or compression of selected flows and not simply on a per packet basis.

6. Implementation and Results

The elements of the IP router on Raw can be classified into four categories as shown in Figure 2: input tiles, route lookup tiles, crossbar tiles, and output tiles. The layout is

```
# neighbors (static network 1)
#define PREV    cSi             # South in
#define IN      cWi             # West in
#define NEXT    cEo             # East out
#define OUT     cNo             # North out

#define ZERO    0
#define TOKEN   0x700

.text
.global begin

# PHASE 1: INITIALIZATION
begin:
  mtsri  SW_PC, %lo(sw_begin)  # load address
  mtsri  SW_FREEZE, ZERO       # unfreeze the switch
  li    $csto, ZERO            # send zero to switch
  li    $csto, TOKEN           # send token
  or    $0, $csti, $0          # receive confirm from switch

# PHASE 2: READ HEADERS
headers_master:
  mtsri  SW_PC, %lo(sw_headers_master)
  mtsri  SW_FREEZE, ZERO
  or     $10, $0, $csti        # receive header (PREV)
  or     $11, $0, $csti        # receive header (IN)

# PHASE 3: CHOOSE CONFIGURATION
  beq    $11, $0, config_master0 # branch on equal
  beq    $11, $5, config_master1
  j      config_master2          # jump

# PHASE 4: PROGRAM THE SWITCH
config_master2:
  mtsri  SW_PC, %lo(sw_config_master2)
  mtsri  SW_FREEZE, ZERO
  addiu  $12, $11, TOKEN       # add immediate
  or     $csto, $0, $12        # send header
  or     $0, $csti, $0         # recv confirm
  j      headers
```

Figure 4. Example of the main processor code.

symmetrical, and all four elements are replicated four times to match the number of ports. Each arrow in the figure corresponds to the data flowing along the static network 1 from one tile to the other in the direction of that arrow. The static network 2, as well as the dynamic network, have not been used in the current version of the algorithm. Due to the fact that the Raw Processor is not physically available yet, the IP router has been tested using the cycle-accurate simulator called *btl* developed in the Raw group.

Following are excerpts from the assembly code to illustrate programming the Raw Processor. Each of the Raw tiles looks very much like a MIPS R4000, and their instruction sets are also similar. Figure 4 shows an example of the main processor code which initializes the switch processor, reads headers, chooses the necessary configuration for the switch processor, and loads the address of that configuration into the program counter of the switch processor.

Figure 5 is an example of the switch processor code which corresponds to the main processor code in Figure 4. PREV, IN, NEXT, and OUT are the directions of previous crossbar tile, input tile, next crossbar tile, and output tile respectively. They are defined in the code above. The "route" instruction connects input ports to output ports of the static network, and also allows communication with the main processor (ports $csto and $csti).

```
.swtext
.global sw_begin

# PHASE 1: INITIALIZATION
sw_begin:
  move   $0, $csto            # receive zero
  move   $1, $csto            # receive token
  nop    route $0->$csti      # send confirm to main
  j      sw_done              # jump

# PHASE 2: READ HEADERS
sw_headers_master:
  move   $2, $PREV route $PREV->$csti # save, send to main
  move   $3, $IN   route $IN->$csti
  j      sw_done

# PHASE 3: EXECUTE CONFIGURATION
sw_config_master2:
  nop   route $csto->$NEXT    # send header from main

  nop   route $IN->$NEXT      # route body
  # ... (unrolled 16 times)
  nop   route $IN->$NEXT

  nop   route $0->$OUT, $0->$csti # send to out, main
  j     sw_done

# DO NOTHING
sw_done:
```

Figure 5. Example of the switch processor code.

As far as the performance results are concerned, the IP router has been tested in five different configurations, each with a different static network packet size of 64, 128, 256, 512, and 1024 bytes long. The only difference between them all was the degree of unrolling of "route" instructions in the switch code to avoid spending extra cycles on branches. Figure 6 shows that it is possible to achieve 76 percent of the theoretical saturation limit of this version of the algorithm with the packet size still within the maximum Ethernet packet size of 1,500 bytes (the theoretical limit is the throughput when sending one 32-bit word every cycle of the processor at the clock speed of 225 MHz). In the simulation, input 1 is sending to output 2 and input 2 to output 3, etc. (one hop clockwise in the crossbar for all inputs), and the system was tested with 1,000 packets total (250 coming from every interface). There are several factors which contribute to this growth of performance, but the most important one of them is certainly the relative amount of time that the static network is kept busy. In other words, in order to achieve better performance of the algorithm it is needed to decrease the processing overhead by spending less relative time in the main processor and more on streaming data through the Raw Processor networks.

The results obtained so far are quite promising, but the Raw Processor can do better, and there are plenty of resources to attain this goal. First of all, there is another static network waiting to be used for routing. As noted beforehand, the second static network can get involved in the algorithm by introducing the second token into the rotating crossbar algorithm. Secondly, while the two static networks

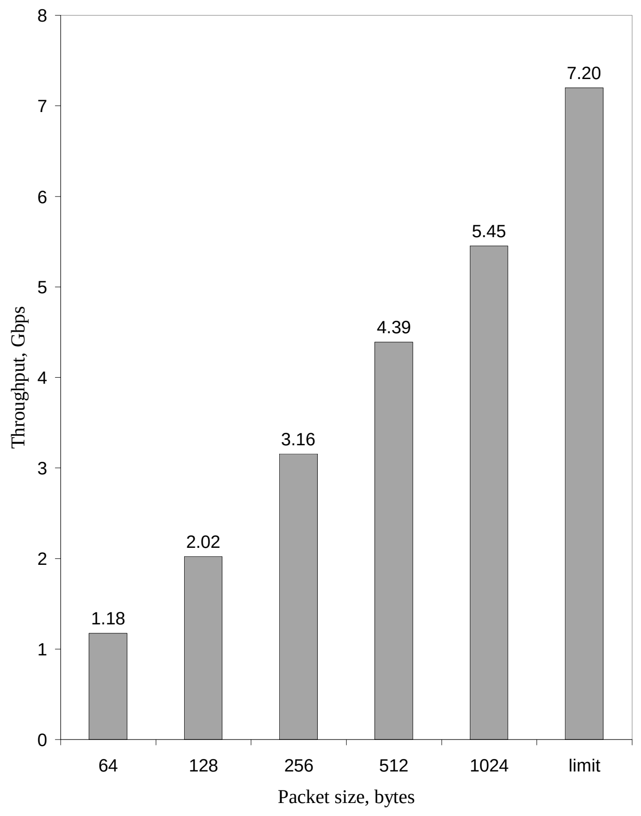

Figure 6. Performance of the IP router.

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. This challenging problem is on the agenda of the authors of this paper.

7. Future Work

The work presented in this paper is presented as a base implementation of a simplistic IP router. In the future, the Raw group would like to implement a more feature filled router. The first thing needed for this will be the implementation of a higher performance interconnection crossbar. The authors know of more efficient, but much more complex algorithms based on the basic idea of a rotating crossbar, that will allow for fair routing on the switch fabric.

Next, the Raw applications team hopes to examine more complicated look-up algorithms with the hope of being able to support enough routes to compete as a core router. To be able to do this, one or several tiles per input port will act as the route resolving entities. One interesting thing to note here is that, usually network processors designed to do this sort of route resolution are natively 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 writeback 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.

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 described in Section 5. This will probably 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. Also, infrastructure for multi-cycle and multi-rate filters needs to be researched.

Lastly, a research direction that holds promise is the application of the computational power of Raw to more intelligent routers. This would be an extension of the Active Networks research thus providing endpoint network users more control over their communications. [6]

7.1. Building Scalable Routers

This paper presented 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. An interesting question that will need to be addressed includes what form of routing algorithm should be used. 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. Another approach that the Raw group is excited about is the use of fair hot potato routing algorithms like the work done in [1].

7.2. Physical Router

The Raw group is a systems oriented group that believes in actually building what it researches. Thus while all of the results presented in this paper have been generated on a cycle accurate simulation, everything about a system cannot be known unless it is completely built. Thus, once the Raw Processor is received back from fabrication, which is to occur summer of 2002, there are plans to build line-card PCB daughter cards. These will plug into the Raw Prototype testboard which has expansion connectors specifically for the purpose of connecting high speed interfaces.

8. Conclusion

This paper has shown that efficient IP routing can be done on Raw's programmable static network. The results obtained in the simulation suggest that it is possible to use the Raw Processor as both a network processor and switch fabric for multi-gigabit IP routing, and mixing computation and communication in a switch fabric lends itself to augmenting the functionality of the IP router with encryption, compression, intrusion detection and other valuable features. The presented rotating crossbar algorithm displays nice properties, such as fairness and scalability, and allows for further improvement by taking advantage of the second static network of the Raw chip. It is also naturally utilizable for implementing Quality of Service. This research is just a starting place for mapping applications to the Raw Processor, and the authors of this paper hope it will stand as an impetus for future work.

Acknowledgments

We thank Walter Lee for his input into this paper. We also thank other members of the Raw group without whose infrastructure and help this paper would not have been possible. This research is funded by DARPA contract # F29601-04-2-0166.

References

[1] C. Busch, M. Herlihy, and R. Wattenhoffer. Routing without flow control. In *Proceedings of the 13th annual ACM Symposium on Parallel Algorithms and Architectures*, July 2001.

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

[3] 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.

[4] W. Thies, M. Karczmarek, M. Gordon, D. Maze, J. Wong, H. Hoffmann, M. Brown, and S. Amarasinghe. Streamit: A language for streaming applications. In *Proceedings of the International Conference of Compiler Construction*, April 2002.

[5] 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.

[6] D. Wetherall, U. Legedza, and J. Guttag. Introducing new internet services: Why and how. *IEEE Network*, July 1998.