EFFICIENT LOOKAHEAD ROUTING AND HEADER COMPRESSION FOR MULTICASTING IN NETWORKS-ON-CHIP

A Thesis

by

POORNACHANDRAN KUMAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Computer Engineering

EFFICIENT LOOKAHEAD ROUTING AND HEADER COMPRESSION FOR MULTICASTING IN NETWORKS-ON-CHIP

A Thesis

by

POORNACHANDRAN KUMAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Eun Jung Kim
	Seong Gwan Choi
Committee Members,	Alex Sprintson
Head of Department,	Costas Georghiades

August 2010

Major Subject: Computer Engineering

ABSTRACT

Efficient Lookahead Routing and Header Compression
for Multicasting in Networks-On-Chip. (August 2010)

Poornachandran Kumar, B.Tech, Indian Institute of Technology Roorkee.

Co–Chairs of Advisory Committee: Dr. Eun Jung Kim
Dr. Seong Gwan Choi

With advancing technology, Chip Multi-processor (CMP) architectures have emerged as a viable solution for designing processors. Networks-On-Chip (NOCs) provide a scalable communication method for CMP architectures with increasing numbers of cores. Although there has been significant research on NOC designs for unicast traffic, the research on the multicast router design is still in its infant stage. Considering that one-to-many (multicast) and one-to-all (broadcast) traffic are more common in CMP applications, it is important to design a router providing efficient multicasting.

In this thesis, a lookahead multicast routing algorithm with limited area overhead is proposed. This lookahead algorithm reduces network latency by removing the need for a separate routing computation (RC) stage. An efficient area optimization technique is put forward to achieve minimal area overhead for the lookahead RC stage. Also, a novel compression scheme is proposed for multicast packet headers to alleviate their big overhead in large networks. Comprehensive simulation results show that with the new route computation logic design and area overhead optimization, providing lookahead routing in the multicast router only costs less than 20% area overhead and this percentage keeps decreasing with larger network sizes. Compared with the basic lookahead routing design, our design can save area by over 50%. With header compression and lookahead multicast routing, the network performance can be improved on an average by 22% for a (16×16) network.

To Parents, God, Family and Friends

ACKNOWLEDGMENTS

I wish to express my profound gratitude to my advisor Dr. Eun Jung Kim for the support and encouragement throughout the research that made this thesis possible. I am grateful to her for having provided me an opportunity to work with her in the field of Networks-On-Chip.

I thank my graduate committee members, Dr. Seong Gwan Choi and Dr. Alex Sprintson for their willingness to serve on my committee.

I cannot forget to thank Lei Wang, a lab mate in the High Performance Computing Lab (HPCL) for having helped me with simulations, ideas, writing and suggestions. I wish to thank all the fellow lab mates and graduate research students of HPCL in the Department of Computer Science and Engineering.

I also would like to thank Ms. Tammy Carda, Department of Electrical Engineering for helping me with various formalities involved in fixing my date of defense, filling application forms and others.

Finally, I thank my parents, family and friends for their loving support and words of encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Evolution of Chip Multi-Processors

Over the past few decades, incredible progress has been made in the field of computers. Computers have become an indispensable part of our lives. Since the first general-purpose electronic computer was created, rapid strides have been made in various aspects like die size, clock speed, performance, storage, compactness, etc. Today a laptop can compute more instructions per second than the ENIACs and big machines of earlier days. Today, we have got external hard disks that store humongous amounts of data in the order of tera-bytes compared to the slow magnetic tapes (almost obsolete now) and much smaller hard disks used before. Moore's law states that the number of transistors that can be placed in an integrated circuit approximately doubles in a space of two years. Microprocessors came into the scene in 1970s and have become an integral part of the computer business because of their ability to ride the big changes and improvements in the computer arena and their cost effectiveness. The performance of microprocessors (nowadays just called processors) have also improved over the years as predicted by the Moore's law.

The exponential performance growth can be attributed only partially to technology advancement. The other important factor that contributed was the introduction of new architectures for processors. During the age of assembly language programming, introducing a new architecture demanded significant effort to ramp up to the new architecture from the old one. But with the emergence of high-level languages and compilers, the assembly language was virtually eliminated. Moreover, the pres-

The journal model is *IEEE Transactions on Automatic Control.*

ence of operating systems like UNIX that are vendor independent made compatibility to a new architecture less expensive [1]. New architectures like RISC (Reduced Instruction Set Computer) were introduced that raised the performance bar with techniques like ILP (Instruction Level Parallelism), Multiple Instruction Issue and cache optimization. CISC (Complex Instruction Set Computer) based processors like Intel began translating their instructions into RISC instructions in order to utilize the techniques provided by them. Though initially the performance improvement was based on technological advances, the introduction of new architectural organization led to about 50% performance growth every year in the 1990s. Minicomputers have been substituted by servers made out of microprocessors. Supercomputers are also now composed of a collection of microprocessors. The performance growth after 2000 faced obstacles like power limits, limit to the ILP, etc.

The major focus of many architecture experts was on increasing parallelism of instruction execution to improve performance. Two of the several techniques proposed were pipelining and multiple instruction issue. Most of the processors today have a pipelined instruction execution. In a pipelined architecture, the instruction execution is divided into various stages like Fetching, Decode, Load/Store, ALU operations, etc. RISC architectures are easy to pipeline because instructions are simple and have a common set of pipeline stages to pass through. While one instruction is being executed in one of the pipeline stages and consuming some specific hardware resource, another instruction can execute simultaneously in any of the other stages assuming the hardware resource corresponding to that stage is currently idle. This leads to parallelism in instruction execution. In general, the pipeline stages are formed such that they consume more or less the same amount of time to complete which is usually one clock cycle in scalar pipelines. Generally, a simple operation like addition of two values present in registers takes one cycle to complete.

However, in what is known as the superpipelined architecture, these pipelines are broken further into smaller stages such that simple operations take m cycles instead of 1 though the result of the simple operation is not available until the m cycles are completed. But these superpipelined architectures can issue instructions each cycle. Thus m instructions are issued before the first instruction completes. Examples of superpipelined processors are CDC 6600 and CRAY-1. The architecture where Multiple Instruction Issue each cycle is implemented is called superscalar architecture [2]. In the superscalar architecture, multiple functional units are present for each operation. This allows multiple instructions to be fetched and executed simultaneously thus increasing the parallelism, in turn increasing the IPS (Instructions Per Second). While superpipelined processors use temporal parallelism, superscalar processors use spatial parallelism. Examples of superscalar processors include Pentium Pro and AMD K5 which decode x86 instructions asynchronously into dynamic microcode instructions and execute them in parallel. Another type of processors that implement ILP is VLIW (Very Large Instruction Word) processors. Unlike the superpipelined and superscalar processors, VLIW processors have an instruction word format that itself stores multiple instructions to be executed. Therefore, the decision of which instructions to be executed in a cycle is made at compile time in a VLIW processor through filling the instruction word, whereas, this decision is made at run time for a superscalar processor. An example of VLIW machine is MultiFlow TRACE [3].

Though it might seem that with more redundant hardware resource available and more the instructions executed in parallel, the higher is the IPS obtained, it is not found to be the case. As the architecture experts were aggressively pursuing more and more parallelism, they very soon hit a roadblock. The parallelism available in majority of the applications seemed to be limited. In the work by Jouppi [4], he said that a machine with a Multiple Instruction Issue of 4 should not be required at all.

He predicted that even for highly parallelizable numeric applications with techniques such as loop unrolling, the non-parallel code limits the total ILP obtained as suggested by the Amdahl's law [1]. Amdahl's law states that if the fraction of parallelizable code is f and the speedup of the parallelizable code is s, then the expression for total speedup is as given in Equation 1.1.

$$OverallSpeedup = \frac{1}{(1-f)+\frac{f}{s}} \tag{1.1}$$

Even with infinite resources when the parallelizable code can be infinitely parallelized, a 10% non-parallelizable code limits the parallelism to 10. The percentage of non-parallelizable code is generally much higher than this number. Hence, it was found from analysis that designing superscalar machines for a parallelism of more than 4 would not be worth the cost of additional resources used in the superscalar machines for most applications.

However, with techniques such as branch prediction [5] [6], trace based execution, speculative execution, the parallelism could be increased depending on how these techniques were implemented. New benchmarks since the work of Jouppi were released which provided better parallelism. But still, the inter-instruction dependencies need to be checked among the instructions to be dispatched which grows in logic as the number of instructions in parallel increases and hence places a limit on the number of dispatched instructions achieved. Also, the applications have their intrinsic dependencies within them. Adding to this the cost of designing a parallel pipeline with redundant resources, the commercial market is yet to see a successful highly parallel superscalar processor. These constraints have shifted the focus onto other alternatives like SMT (Simultaneous Multithreading [7]), EPIC (Explicitly Parallel Instruction Computing [8]), and multi-core processors. SMT deals with executing instructions from multiple threads in the same cycle. It requires a superscalar architecture to do so. While a normal superscalar processor executes parallel instructions without considering which thread they come from, SMT executes instructions from different threads. But SMT suffers from certain security issues and may slow down some applications if not optimized properly. There is not enough OS (Operating System) support to determine if the SMT is going to be advantageous or detrimental to an application. EPIC is based on VLIW machines and focuses on shifting the instruction issue from the hardware to the compiler. Again, VLIW has the problem of variable instruction formats and incompatibility with previous architectures. Multi-core processors on the other hand are two or more independent cores put together on a single integrated circuit die or chip. They are also known as CMPs (Chip Multi-Processors). CMPs have always existed in the past but until recently they were always considered to belong to the high-end exotic market rather than the mainstream market where uniprocessors were dominating. But uniprocessors reached a saturation point because of various reasons. Clock speed could not be increased any further because of power concerns like chip over-heating and memory speed limitations. Moreover, superscalar processors, superpipelined processors and others described above have a high design complexity and are expensive to build [9]. On the other hand, CMPs can be built with scalar uniprocessors that have become almost a commodity in the market. Also, with rapid progress in the die technology, it is easier and affordable to fit in more cores than ever before. Also, a CMP of superscalar processors can be built if necessary. Typically, CMPs have multiple cores with each core executing one thread. With such flexibility, CMPs have become the most attractive option for computer architects.

B. Communication Using Networks-On-Chip

CMPs are widely used in todays machines. The multiple cores inside a CMP need a way to communicate with each other. This has been provided till now by a common bus. A bus typically connects all the cores and provides a communication medium among them. Modern buses can be used either in a parallel connection or in a bit serial mode. Buses range from memory-CPU connector buses to the highly advanced HyperTransport and Infiniband [10] buses. HyperTransport (HT) is a DDR (Double Data Rate) bus that is used for various applications in a computer system like Front Side Bus Replacement, Multiprocessor Interconnect, etc. DDR means that HT transmits data on both the rising and falling edges of the clock. HT is used by AMD for its Opteron processor. The Opteron processor maintains coherency among its caches by a cache coherence protocol that is based on HT [11]. When used in full bandwidth mode, HT is faster than most bus standards for High Performance Computing. Infiniband is a third generation bus that is based on a switched fabric communications interface. It provides QoS (Quality of Service) and has the capability of failover (i.e., to switch automatically to a backup network in case of failure). Infiniband is designed to be scalable though most of the buses are not scalable. With growing number of cores in CMPs, a scalable communication interface among the cores is required.

A scalable interface is one where adding more cores doesnt overload or saturate it very soon. Let us consider the scenario of CMPs. Memory architecture in a CMP can be classified into shared memory architecture and distributed memory architecture. In a shared memory architecture, a single memory is accessed by multiple processors (cores) inside the CMP. In a distributed memory architecture, the processors have a local private memory and local shared memory which can be shared with other processors. In both shared and distributed memory, processors send requests to memory for data and stores the received data in its cache. Once data is stored in caches, multiple copies might exist for the same data. Therefore, when a processor sends a request for that data on the network, there should be a way to know if the most updated copy is present in the memory or in one of the caches. To maintain consistency among the copies of data, there are different cache coherence protocols that have been employed. The cache coherence protocols can be broadly classified into two categories namely snoop-based cache coherence protocols and directory-based cache coherence protocols. Snoop-based cache coherence protocols basically broadcast the memory request placed by a processor on a network. Each of the other processors replies back if it has a copy of that data or not and whether that copy is more updated than that present in the memory. A directory based cache coherence protocol relies on the memory to send and gather information about redundant copies in various processor caches. This type of interaction to maintain cache coherency requires broadcasts among various processor cores. Moreover, the cores place requests frequently on the network thus creating heavy traffic on the network. A common bus cannot satisfy such bandwidth requirements beyond a limited number of cores. Thus, point to point network links are required to support scalability. Such point to point links constitute a NOC (Network-On-Chip) because they form a network and are integrated on the die.

NOCs come in various configurations, topologies and sizes. They are packet based and can employ different routing algorithms to transfer packets from a source to a destination. NOC employs flow control to avoid deadlocks and livelocks. Thus NOC has emerged to be a scalable, point to point network interface that can be used for efficient communication among the cores. The three important components of an NOC are routers, links, and NICs (Network Interface Controllers). Links connect one router to another or a router and a NIC. They transfer packets from one end to the other. Routers are attached to two or more links and receive incoming packets from one link and forward onto another link. Routers make the decisions on which link to forward the received packet based on the final destination of the packet. Thus the router needs a Routing Computation stage that analyzes the packet and determines the destination of the packet. The router has a crossbar switch so that the packet can switch into the right output link for forwarding. To make this switching decision, the router has a switch allocator that allocates an appropriate switch available for the packet transfer. If the router is busy with some other packet, then the current packet has to be stored so that it is not dropped before the router is available to process this packet. Therefore, the router has enough buffers to store incoming packets.

NOC can be built as a ring network where the processors are connected in a chain or ring fashion [12]. The processors can be connected in a grid to make a 2-Dimensional (2-D) Mesh network [13] [14]. Fig. 1 shows a 2-D mesh network and a ring topology. As can be seen from Fig. 1, the ring topology has a drawback that any packet transfer will take an average of N/4 hops (where N is the number of processors) provided the links are bidirectional. On the other hand, for a $N \times N$ 2-D mesh network, we can transfer a packet from one corner to a diagonally opposite corner in 2N-2 hops. Therefore, a ring network is not as scalable as a 2-D mesh network. For smaller number of processors, ring topologies are easier to implement than a 2-D mesh. Also, ring topologies have an inherent broadcast capability that can be exploited for coherency protocols that typically involve a lot of broadcasts.

C. Importance of Multicasting

CMPs with NOC integrated into them have become a major focus of research for the implementation of multiprocessors. One of the major areas where such systems are

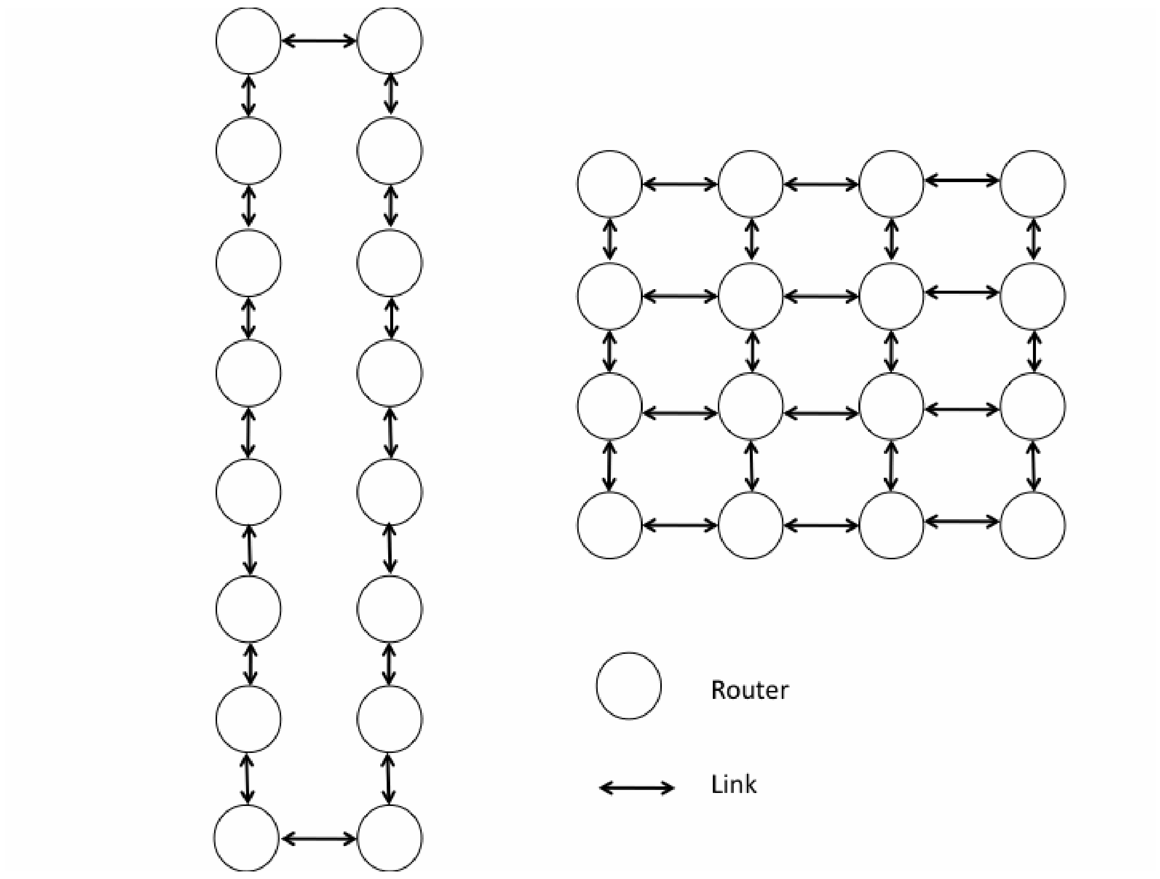


Fig. 1. Sixteen node ring topology and 16 node 4x4 2-D mesh topology.

used is shared memory multiprocessors. With a large number of cores and associated caches, there is a lot of traffic generated by the coherence protocol. As discussed earlier, many of the message packets generated have to be sent to all the cores in the network in case of a snoop-based coherence protocol [15]. Even for a directory based cache coherence protocol [16], a lot of packets have multiple destinations. Sometimes this traffic can create a lot of network congestion and network resource depletion. Therefore all measures need to be taken to minimize the number of links used to transmit the packets to the destinations . One of the most effective methods to do this is multicasting [17] [18].

Multicasting, in comparison to broadcasting means to send a packet to multiple destinations instead of all destinations as in the case of broadcasting. Another approach called multiple unicast is to send one packet replica to each destination in the destination list of the packet serially. This is inefficient when compared to multicasting. Multicasting takes advantage of the common links along the paths to the different destinations. If there are a cluster of destination nodes that have a common path from the source node up to a certain intermediate router, then multicasting sends only one packet to that intermediate router which then replicates the packet and transmits one of those in every different direction the destinations are present. This process is repeated until the packets reach all the destinations. On the other hand, multiple unicast determines the path to each destination and sends a packet to it. Thus the common links along the paths are used again and again leading to high bandwidth usage and additional power consumption. Multicasting is more advantageous than broadcasting in huge networks with point to point links, especially for cache coherence protocols. The reason is that, in many cache coherence protocols, the list of nodes that have a copy of a particular data is known and is a small percentage of the total number of nodes. Therefore, it is pointless to broadcast the messages to all the nodes consuming extra bandwidth and power and also reducing throughput. Instead, multicasting can be used to send the coherence messages to only those nodes that hold a copy. Also, multicasting can be used to implement broadcasting efficiently without repeated use of common links. Fig. 2 shows through an example the difference between Multicasting and Multiple Unicasting. The links AB, BC and CD are used twice in the case of Multiple Unicasting while they are used just once in Multicasting. Hence, link resource utilization is reduced in Multicasting thereby reducing power and link bandwidth utilization.

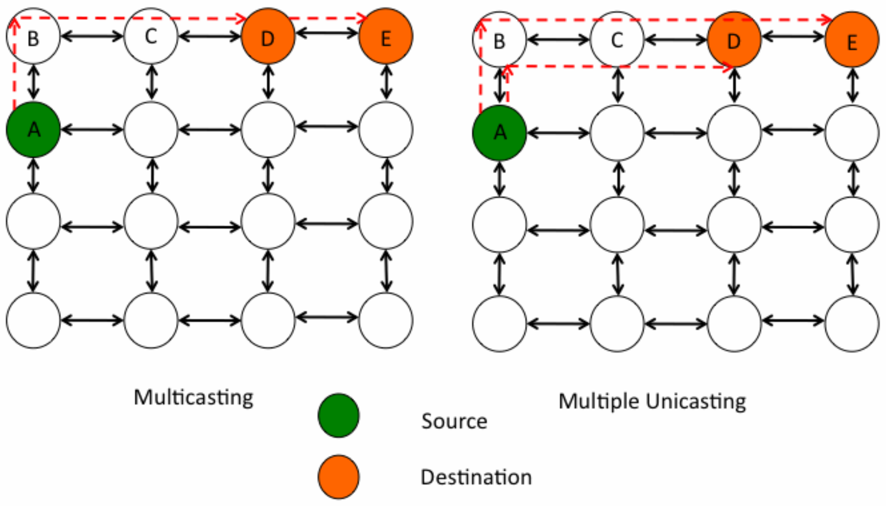


Fig. 2. Comparison of multicasting against multiple unicasting. Multiple unicasting utilizes almost double the link resources used by multicasting in this example.

D. Compression Schemes

Compression is an old technique and has long been used in the field of communication ranging from audio, video compression to data compression. However, compression schemes are new to CMPs and NOCs. Compression schemes are hard to implement inside the routers because of the overhead in terms of hardware and latency. Therefore, the highly sophisticated and complex compression schemes that are used in other fields are difficult to integrate in NOCs. Hence, most of the compression schemes that have been specified for NOCs are implemented inside the NIC or inside the processor itself. Another reason for implementing compression schemes at the NOC interface is that the payload data does not undergo any change inside the network. Compression schemes have been found to be important in reducing communication latency thereby enabling performance improvement. Compression schemes also help in power reduction by consuming lesser network resources. Some of the previously proposed compression techniques for NOCs are USBR (Unused Significant Bit Removal [19]), Adaptive Cache Compression for High-Performance Processors [20], cache compression and NIC compression using frequent value patterns, Frequent value locality and value centric data cache design [21], etc.

CHAPTER II

BACKGROUND

A. RPM – Recursive Partitioning Multicast

The work of this thesis is based upon a multicast routing algorithm called Recursive Partitioning Multicast (RPM). The RPM scheme is described in detail in [22]. RPM deals with the general multicast problem of where to replicate packets in the network. Poor replication decisions can lead to high power consumption and substantial degradation in the network performance. RPM uses a bandwidth efficient algorithm to make decisions on where to replicate the packets. Fig. 3 shows an illustration of how decisions on where to replicate the network can impact link congestion and bandwidth consumption in the network. The example shown in Fig. 3 is a 4x4 mesh network. We assume router 5 to be the source and routers 3, 7, 11 and 15 as destinations. In the network on the left, replication is done at routers 5 and 9. Replication is done at routers 7 and 11 for the network on the right . Even though the number of replications (which is 3) is the same in both the cases, the left network consumes 11 links while the right consumes just 5 links. This certainly shows that intelligent decisions can reduce bandwidth and power consumption of the network. RPM is one such intelligent scheme for multicast routing.

The basic idea of RPM is to compute a routing path for a packet based on the locations of routers present in the destination list for that packet. The network is partitioned at the current router and then based on these partitions and the destination list for a packet, the output ports to send the packet to are determined. The network makes one replica with an updated destination list for each output port. These updates make sure that each destination is present in the destination list of only one

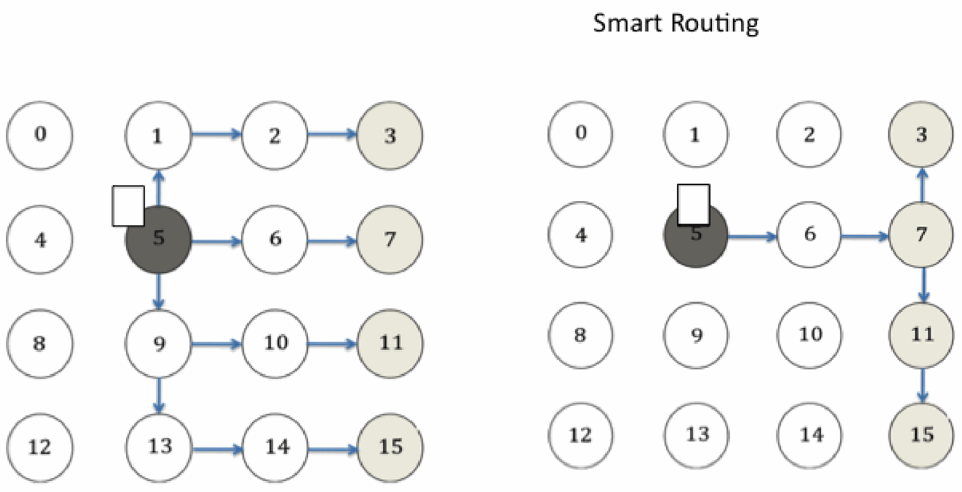


Fig. 3. Two different ways of multicast routing for a fixed set of source and destinations.

packet replica thus preventing redundant packet delivery. The RPM logic consists of two steps. The first step is to find out which parts of the network the packet needs to be sent to. The network is divided at the current router into 8 partitions using a partitioning logic and the bit encoded destination list is taken from the header and fed as input to the partitioning logic. Eight signals, one for each partition, are generated which indicate if that partition contains any of the destinations from the destination list. The second step is to make routing decisions. The signals generated in the first step are used by the routing algorithm and the decision on which ports of the current router (N, E, S, W) to send the packet to is made. Algorithm 1. shows the pseudo-code for the implementation of the second step. The pseudo-code to implement the routing decision making logic is based on the following priority rules:

- North has a greater priority than East to reach part 0 (Northeast) destinations.
- West has a greater priority than North to reach part 2 (Northwest) destinations.
- South has a greater priority than West to reach part 4 (Southwest) destinations.
- East has a greater priority than South to reach part 6 (Southeast) destinations.
- If there are destinations in both part 0 and part 2, North is used for both these partitions. Similarly, South is used if destinations are in both part 4 and part 6. If part 1 and part 2 have destinations but not part 3, then North is used for both part 1 and part 2. Similar rule is applicable to the South direction.

The reasons for choosing RPM as the base multicast algorithm are manifold. Several multicast routing algorithms exist as of today ranging from Multiple Unicast to recent works like VCTM (Virtual Circuit Tree Multicasting [23])and bLBDR [24], which is an extension of LBDR (Logic-Based Distributed Routing). VCTM suffers

Algorithm 1 Pseudo Code for Routing Computation [22].

```
1: if IN(dest,7) OR (IN(dest,6) AND !IN(dest,5) AND !IN(dest,4)) then
2:     ADD(EAST)
3: end if
4: if IN(dest,1) OR (IN(dest,0) AND (!IN(dest,7) OR !IN(dest,4) AND IN(dest,6)))
   OR (IN(dest,0) AND IN(dest, 2)) then
5:     ADD(NORTH)
6: end if
7: if IN(dest,3) OR (IN(dest,2) AND !IN(dest,1) AND !IN(dest,0)) then
8:     ADD(WEST)
9: end if
10: if IN(dest,5) OR (IN(dest,4) AND (!IN(dest,3) OR !IN(dest,0) AND IN(dest,4)))
   OR (IN(dest,4) AND IN(dest,6)) then
11:     ADD(SOUTH)
12: end if
```

IN(dest,n): at least one destination node is in part n.

ADD(direction): add direction as candidate.

from additional storage overhead for managing the tree information for multicast. VCTM requires added multicast latency to send a setup packet to build a tree. VCTM is also not scalable for large networks because every time a source node changes, it needs to set up the multicast tree once again. bLBDR doesn't work well in networks where destinations are spread out far and wide. On the other hand, RPM is scalable, doesn't add any additional latency and doesn't require additional storage requirement. RPM needs only combinational logic to implement the partitioning algorithm. For small networks, the area of this partitioning logic is negligible.

A clear understanding of how the partitions are created can be obtained from the following example. Consider a 4×4 network where the nodes are named from node 0-15 as shown in Fig. 4(a). The source node is node 9 and the destination nodes are nodes 0, 2, 3, 13 and 15. The source node 9 is the current node at the beginning. The partitions are as shown in Fig. 4(a). According to the priority rules, since destination 0 is in part 2 and destinations 2 and 3 are in part 0, north direction is used to reach these three nodes. Also, since node 13 is in part 5 and node 15 is in part 6 but there is no destination in part 7, south direction is used to reach node 13 and node 15. Therefore node 9 replicates the packet once and sends it to node 1 in the north and node 13 in the south. Node 1 partitions the network and looks at the destinations in it. The destinations are to its east and west. Node 1 then replicates the packet and sends it to nodes 0 and 2. Node 0 sends it to its ejection port while node 2 replicates the packet and sends it to its ejection port and to node 3. In the south, node 13 partitions and sees that the destinations are to its east. Hence, it replicates and sends one to its ejection port since it is also one of the destinations and another along its East direction to node 14. Node 14 again partitions and looks at the destinations to send to. There is only one destination which is towards its east direction. Hence it forwards the packet to node 15 which then forwards it to its ejection port.

Next, we look at the hardware implementation of the RPM routing logic which is shown in Fig. 5 (Courtesy [22]). The partitioning block takes the destination bit vector as input and computes the eight signals for each partition as discussed before. The logic outside the partitioning logic block represents the hardware implementation of the pseudo code shown in Algorithm 1. The input to the routing decision making logic are the eight signals part 0 - part 7. The partitioning logic block for node 9 (Fig. 4) can be conceived in hardware as illustrated in Fig. 6.

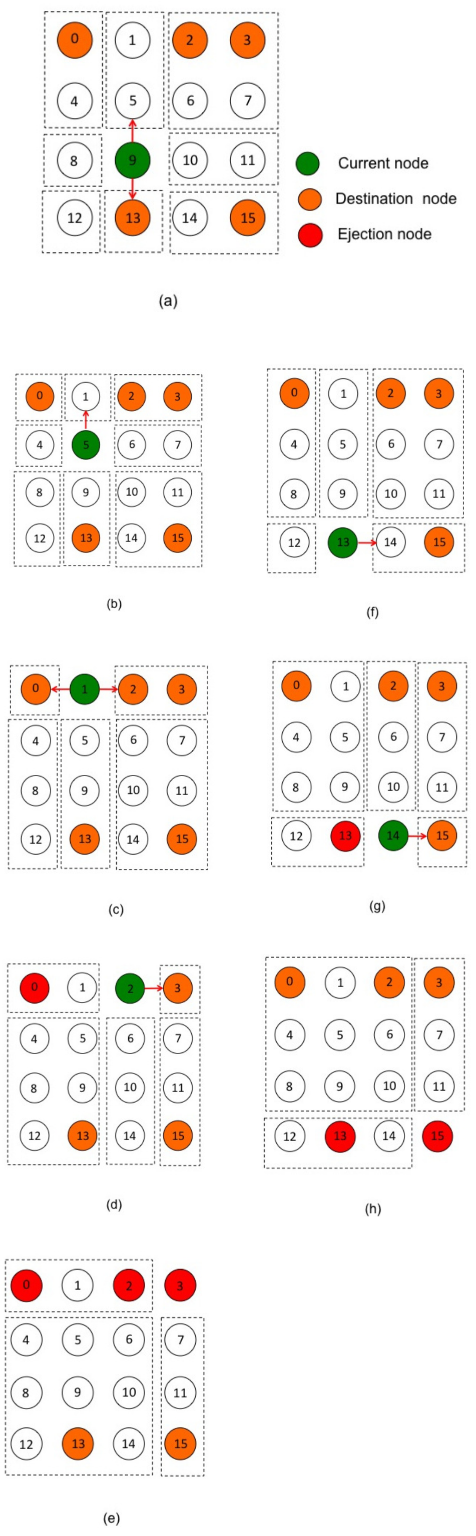


Fig. 4. Multicast packet traveling example.

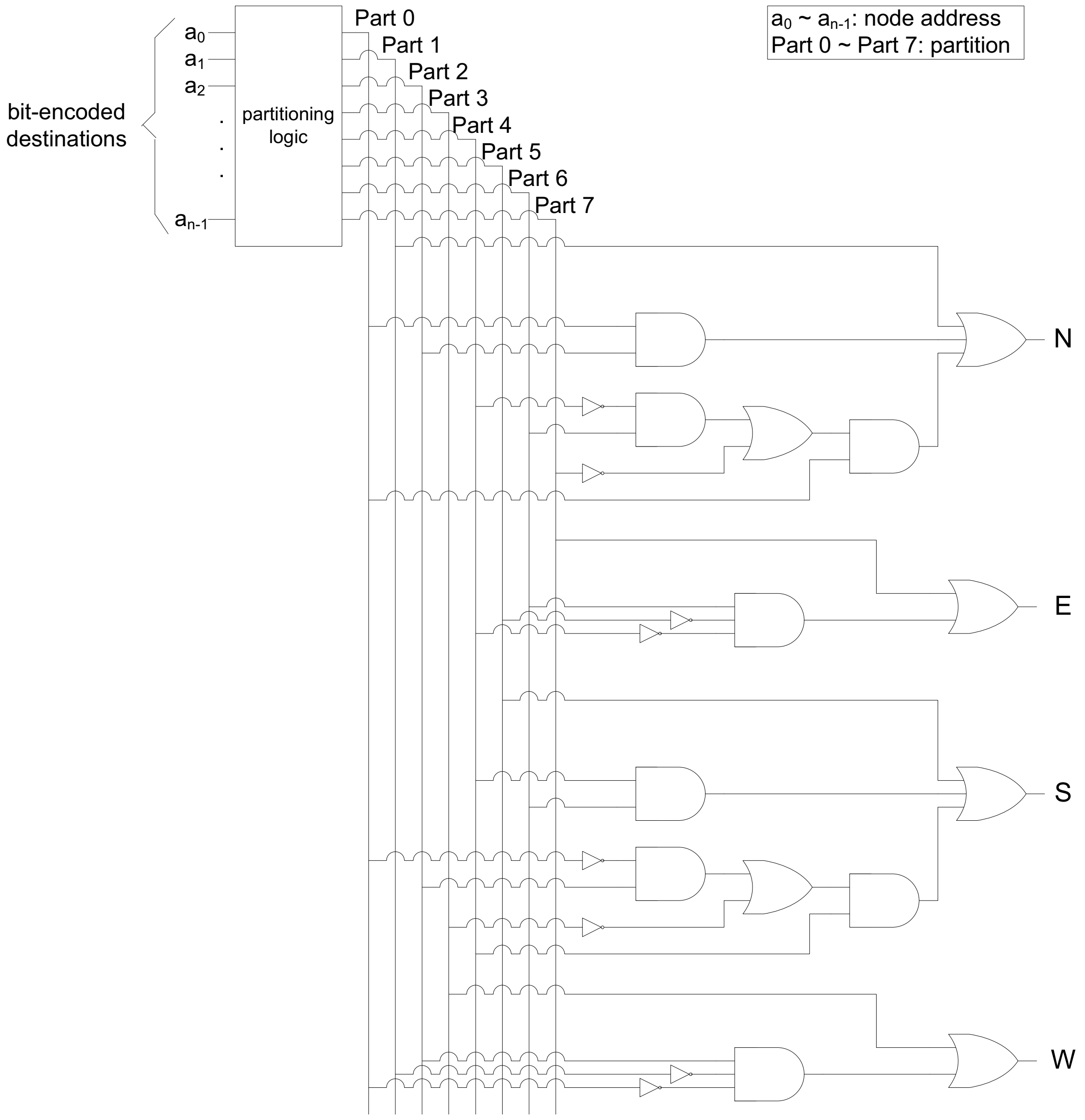


Fig. 5. Hardware implementation of routing logic.

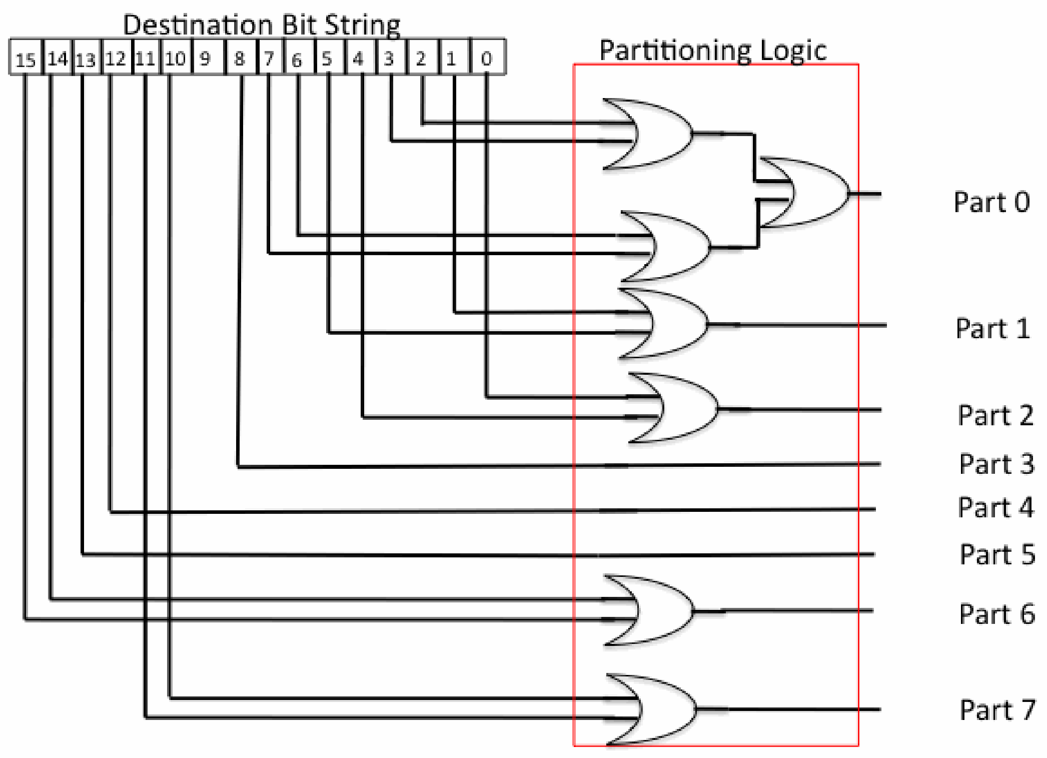


Fig. 6. Partitioning logic inside router 9 for the (4×4) network shown in Fig. 4(a).

CHAPTER III

RELATED WORK AND MOTIVATION

A. Lookahead Routing

The block diagram of a typical baseline Virtual Channel (VC) router employed in NOCs is shown in Fig. 7 [25]. Based on the functionality, the blocks can be classified into two groups - the datapath and the control path. The datapath takes the responsibility of handling the storage and movement of the packet's payload. The datapath consists of the input buffers and the switch. The control path is comprised of the remaining blocks namely the Routing Computation block, the VC Allocator (VA) and the Switch Allocator (SA). The control path coordinates the movement of packets through the datapath components. The tasks performed by the control path include Route Computation, VC Allocation and Switch Allocation. These functions are explained later in this section.

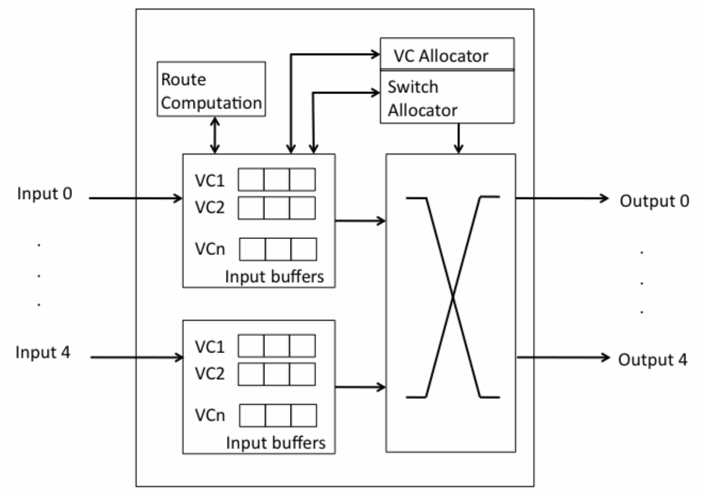


Fig. 7. Baseline router architecture.

The router assumes that worm-hole routing is enabled. Worm-hole routing or Worm-hole switching is a form of flit-buffer flow control. Packets are divided into various chunks called flits (flow control digits). The first flit is called the header flit and the last flit is called the tail flit. The header flit holds the route information of the packet (like the destination address). Between the head and the tail flits exist zero or more body flits also known as middle flits. The middle flits contain the actual payload while the tail flit contains information to close the connection with the router i.e., to release router resources after the packet has been transmitted. The middle flits and the tail flits blindly follow the head flit on its route. Flow Control decides when these flits move forward from a router. Worm-hole flow control implements virtual channels and is therefore suited for the router architecture shown in Fig. 7. In worm-hole routing, the head flit of a packet doesn't wait for the entire packet to arrive to move forward to the next router. Instead, each flit moves independently but along the same route and virtual channel. If a packet has b flits and the packets route has a distance of d hops, then the time the packet takes to reach the destination is d+b-1.

Without a virtual channel, the full link is allocated to the flits of a packet which might result in blocking of the channel in case of contention. If a flit is not able to move forward, all the links occupied by the packet flits are now blocked. To avoid this, a link is sub-divided into many virtual channels and a virtual channel is allocated to each packet at a router. Thus, other packets have other virtual channels to move ahead and are not blocked by this packet. The virtual channel holds the necessary state to manage the flits of a packet. For instance, the virtual channel knows the output channel for the next hop of the packet route. The virtual channel also keeps track of the state of the output channel (if it is busy, or idle). The name worm comes from the fact that the flits of a packet may occupy many flit buffers along the path of the packet thus looking like a worm.

The input unit of a router contains flit buffers that store incoming flits from packets. The flits are stored until they can be forwarded to the next router. Also, the state of every virtual channel associated with that input link is maintained in the input unit. To advance the packet to the next router, the control path begins with the task of Route Computation. In this stage, the output port to which the packet should be forwarded is determined. Given the output port, the packet then places a request to obtain an output virtual channel. The output virtual channel is allocated by the VC allocator. Once the output port and output virtual channel are allocated, the switch allocator takes control and allocates a time slot for the packet to traverse to the output channel. Some router architectures have output buffers to store the flits in the case of unavailability of input buffers in the next router. But, the design mentioned in Fig. 7 doesn't have any output buffers. Hence, prior to forwarding the flit, it is made sure that there are empty flit buffers in the next router corresponding to that output channel.

Fig. 8 shows the router pipeline of a typical virtual channel router. Each flit must pass through the stages of Route Computation, VC Allocation, Switch Allocation, and Switch Traversal to get forwarded to the next stage. Fig. 8 reflects a router that takes one clock cycle for each of the pipeline stages. Moreover, there are no stalls at any stage in the pipeline. During cycle 0 (not shown in the figure), the head flit arrives at the router. Based on the information in the head flit, the router is directed to a specific virtual channel. During cycle 1, the RC stage reads the head flit and determines the output port to route the packet. At the end of this stage, the virtual channel is waiting to get an output virtual channel. During cycle 2, the VC allocator requests an idle output virtual channel for the head flit. In the meanwhile, the first body flit (Middle flit 1) passes through the RC stage. In the VA stage, the VC allocator takes as input the output channel that was determined in the RC stage and finds if a virtual channel in that output link is available. If available, the VA successfully allocates that output virtual channel for the current packet and this is stored in the state of the input virtual channel until the packet is completely transmitted.

During cycle 3, Middle flit 2 enters RC stage, Middle flit 1 enters the VA stage and the head flit enters the SA or the switch allocation stage. In the switch allocation stage, the packets are handled at the individual flit level because the packet processing has been done and an output virtual channel has been allocated. All the packet has to do now is to grab a switch and traverse to the output channel. Therefore, the header flit is treated no differently than the other flits. Switch allocation is now done for each flit. Each virtual channel that has buffered flits with output virtual channel and downstream buffer available bids for a single flit time slot. If the switch bid is successful, the flit traverses the switch and into the output virtual channel and gets stored in the downstream buffer. A credit system is maintained to keep track of downstream buffer availability. If a credit is available, it means that there is a downstream buffer that is empty. When a flit traverses the switch and ends up in the downstream buffer, the number of credits is decremented by 1. Similarly, the other flits also obtain a single flit time slot and traverse the switch and onto the output virtual channel to get stored in the buffer of the next router upon which a credit is decremented. During cycle 8, the tail flit enters the switch traversal stage after grabbing a single flit time slot. Once the tail flit is transmitted, the input virtual channel is now released.

However, the timeline of the router stages depicted in Fig. 8 assumes that all the bids for VC and switch allocation were successful. But this is not always true. In reality, when multiple packets bid for the same virtual channel or switch, only

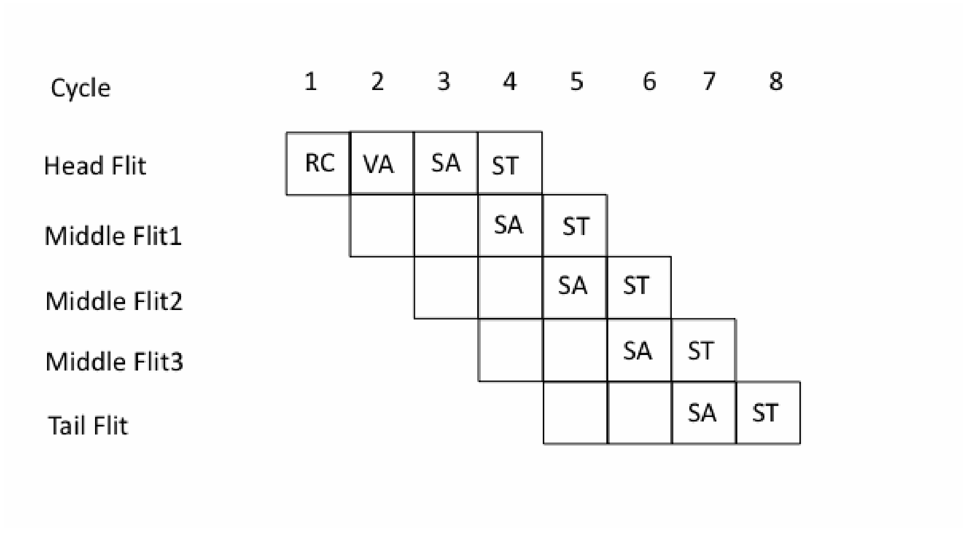


Fig. 8. Virtual channel router pipeline in action for packet routing.

one is bound to be successful in that cycle and rest of the packets have to try again. Therefore, stalls occur whenever there is an unsuccessful bid. Stalls are those that occur because the router doesnt do anything with a flit in that cycle because of lack of availability of resources. An example of a stalled router pipeline for the same packet as in Fig. 8 is shown in Fig. 9.

The stalls in the pipeline increase the transfer latency of the packet. Work has been done to minimize the router pipeline through concepts known as speculation and lookahead because latency of the interconnection network is directly related to the depth of the pipeline. Speculation aids in executing two stages in parallel which otherwise would be sequentially executed. For instance, VA and the SA stages were done sequentially because only after the virtual channel has been set up, the switch allocation could be done. But with speculation, it is assumed that VA stage bid will be successful and therefore SA stage could be done in this cycle. When the VA and SA stages are executed in parallel in one cycle, either VA or SA or both VA and SA stages can fail. If VA fails and SA is successful, SA is discarded and in the next

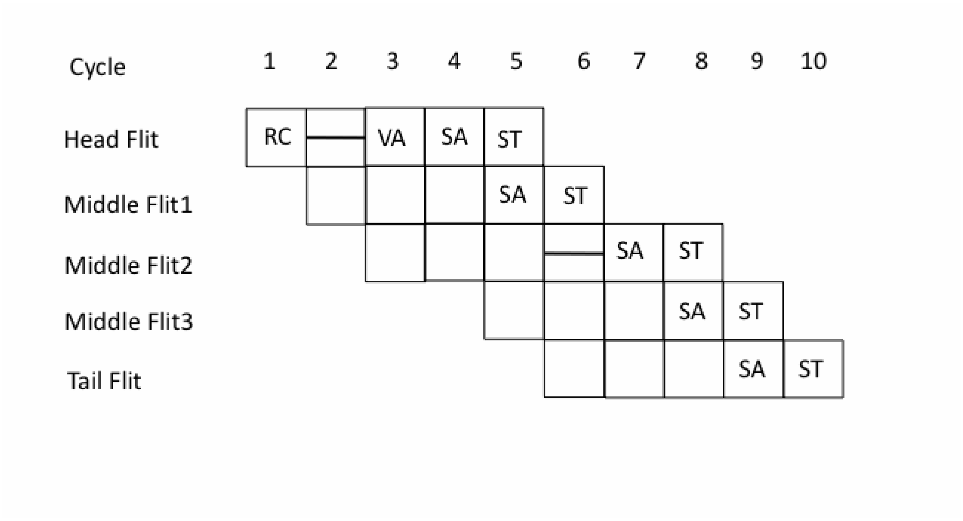


Fig. 9. Two types of stalls in router pipeline. The head flit is stalled in cycle 2 because the VC allocator bid was unsuccessful. This is an example of a packet stall since it is related to packet processing. The middle flit 2 also encounters a stall in cycle 6 which is a flit stall since by now the channel for packet transfer has been established by the router.

cycle, VA and SA are done again. If only SA fails, then only SA can be done in the next cycle. If both fail, again both VA and SA stages are executed in the next cycle. Sometimes with speculation, the router pipeline can be reduced to a two stage pipeline with ST stage also combined with VA and SA as shown in Fig. 10.

While speculation can be done to combine VA, SA and ST into one cycle, it cannot be used to combine RC with other stages. This is because without RC done, we dont have an idea of where the packet is going. Therefore, there is no use in bidding for any virtual channel of any output port. But the work of the RC stage for a packet can be done prior to its arrival in a router. Once the VA stage is done for a packet, the router can compute RC for the next packet and store the result along with the head flit which can be used directly by the next hop router. This concept is called lookahead routing and is employed in many NOC routers where reducing

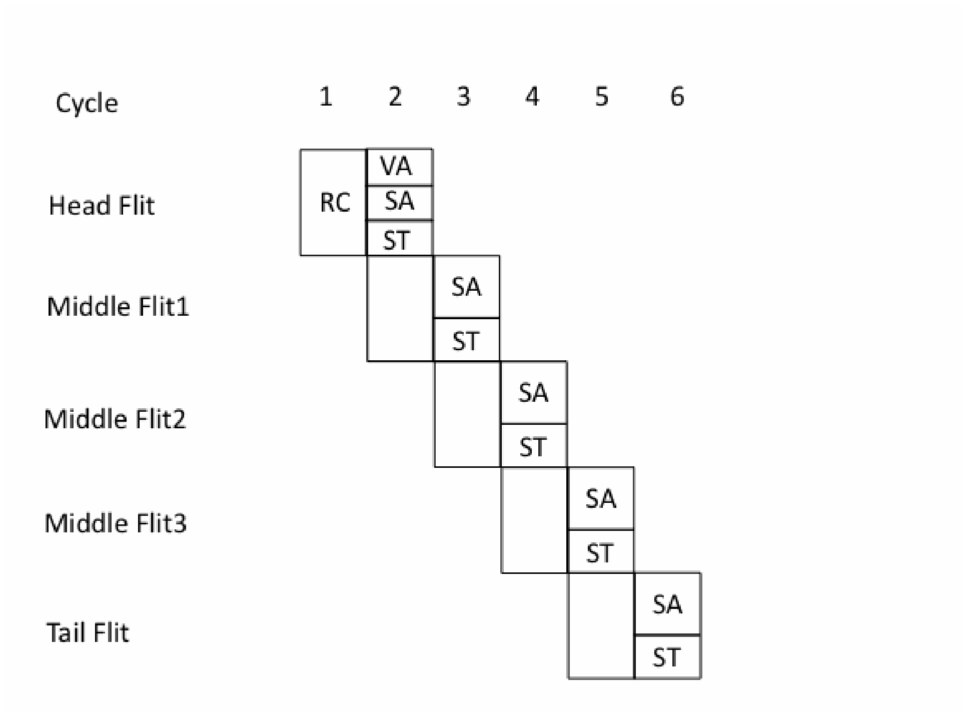


Fig. 10. Speculative router pipeline with stages VA, SA and ST executed in parallel.

latency has a high priority. This RC computation can be overlapped with the VA or SA stage or both stages without loss of performance. A lookahead pipeline is shown in Fig. 11.

So far, lookahead routing was discussed in the context of unicast processors in which only one next hop route is present and thus calculating lookahead route is pretty simple. The RC stage that was present in the non-lookahead router can be used for this purpose and thus there is no extra hardware added. But, when multicast routers are considered, many lookahead routes need to be computed for each next hop router to which the packet is to be sent. Therefore the question arises as to how much additional area and logic is required to take care of lookahead routing. A lot of multicast schemes have been proposed till now but none have considered lookahead routing in detail. Two of the most recent works on multicasting are Virtual Circuit

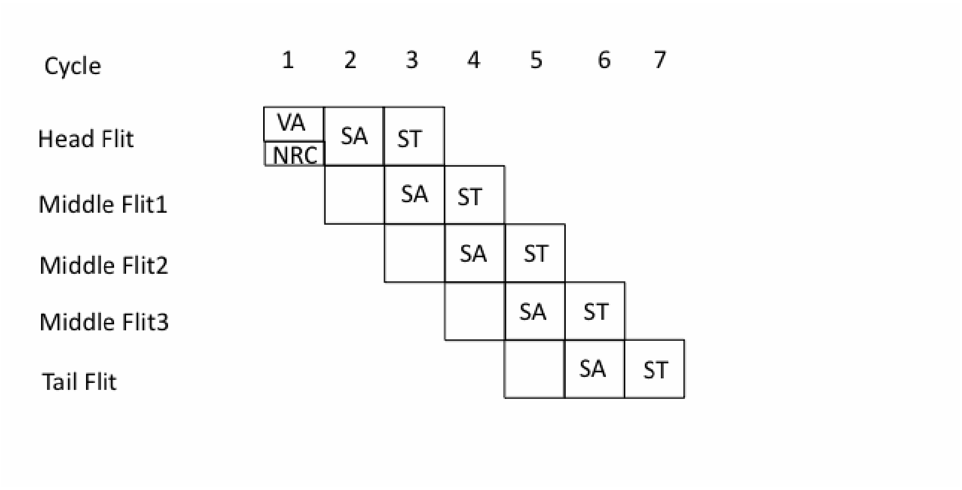


Fig. 11. Pipeline with lookahead routing. NRC stands for Next Route Compute and is overlapped with VA stage.

Tree Multicasting (VCTM [23]) and Recursive Partitioning Multicast (RPM [22]). These two schemes implement lookahead routing but don't mention about the area overhead due to lookahead routing.

As discussed in the Chapter II, VCTM supports multicast by having a Virtual Circuit Tree table stored in each of the routers. If the circuit tree is not present, it is built by sending a unicast to individual destinations of the multicast packet thus adding the path to the tree. Once the circuit is set up, all that needs to be done is to look up the VCT table and send packets to the output ports listed in the table for that multicast tree. The lookahead router pipeline used in VCTM is illustrated in [23]. The RC computation is done one hop ahead for the unicast packets and the VCT table look up is done a hop ahead for multicast packets. For multicasting, depending on the number of branches in the router, the packet is replicated that many number of times. For each replication, the VA and SA stages are executed one by one and the packets transmitted to their respective output ports. Coming to the VCT implementation for lookahead routing, let us first consider the VCT table without lookahead. A typical VCT entry has rows of entries where each entry maps to a particular Virtual Circuit Tree. Each entry is a bitmap of the output ports to send the packet and the last column of the entry lists the total number of branches from this router. Thus looking at an entry, the router determines the output ports to forward the packet.

In case of VCTM multicast with lookahead routing, the router has to determine which output ports of the next hop routers the packet must be forwarded to. Multiple next hop routers may be present and they might have the same output port (Eg. East output port) to forward to. Therefore, to avoid overlapping of same output ports from different next hop routers, it is necessary to have the bitmap for the output ports for each next-hop router separately. For a 2-D mesh, this increases the size of the VCT up to four times. Looking at the area estimates of the VCT tables for 4096 entries in the area and power analysis section, this is a significant increase in the area overhead. Such a magnitude of area overhead might also affect the scalability of the VCTM scheme.

It can be argued that there is no need to store the output port bitmap of each next hop router separately. The only way to implement the VCT then is to have the same table as that for no lookahead routing and allow overlapping of output ports for different next hop routers onto the bitmap. In that case, there is no way to know which router a bitmap of an output port is set for. For example, if an entry contains the bits corresponding to E, W, and S set and the number of next hop routers is three, there is no way to know which of the three next hop routers was the East Output port associated to and similarly with the West and South directions. Therefore, the only possible solution might be to send the packet at each next hop router in all directions listed in the VCT table entry. This might create branches that didn't exist in the VCT tree at all. This might lead to an increase in network congestion. Also,

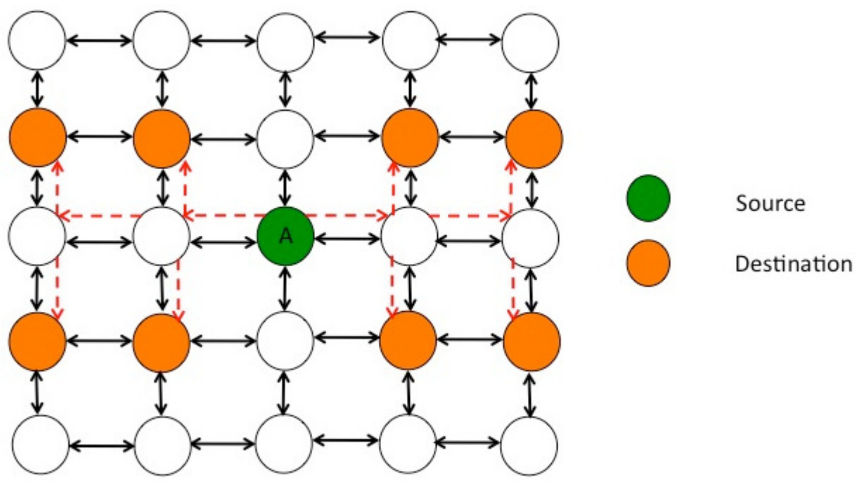


Fig. 12. Virtual circuit tree setup for the destination nodes.

this might result in two further consequences. One is that, because of the misleading bitmaps, some packets might be sent to routers that are not part of the VCT tree. So, when a multicast packet enters such a router, it tries to look up the VCT table thinking there should be an entry for it already setup but doesn't find one. Hence, if no dropping mechanism is employed, the packet sits there forever not knowing where to go next. Another consequence is the possibility of repeated exchange of packets between adjacent routers without any additional scheme to avoid this. Consider the network that has been setup for VCTM lookahead multicast as shown in Fig. 12. The network is a 5×5 2-D mesh network with the source being A and the destination nodes colored in saffron.

Unicast packets are sent to each destination individually and finally the multicast tree is setup as shown by the dashed lines in red. VCTM assumes XY routing and hence the path is as obtained as shown in Fig. 12. Fig. 13 shows the entry for the lookahead multicast tree in Fig. 12. Since the router to the left of A has to send along

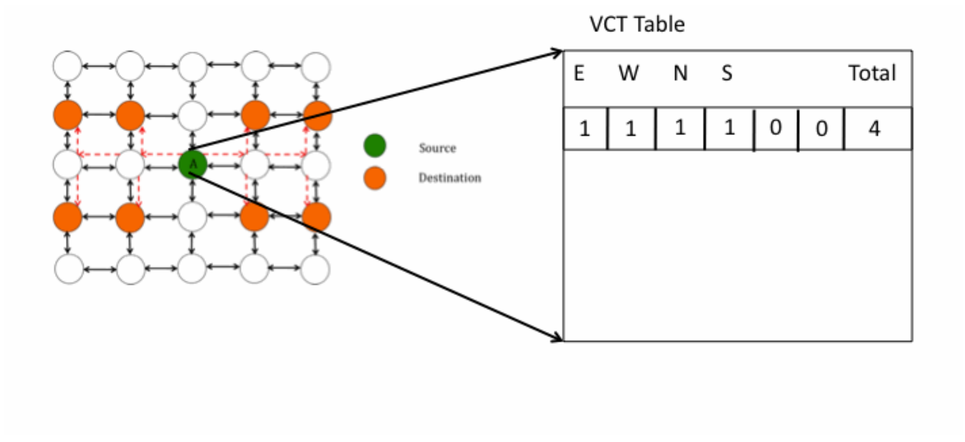


Fig. 13. Lookahead VCT table at router A storing the overlapped bitmap for its next hop routers.

North, South and West output ports, and the router to the right of A has to send along North, South and East output ports, according to the lookahead scheme being considered, the VCT table in A contains the bits for North, East, West and South all set to 1. But there is no way to know whether West output port set to 1 was associated to the router left of A or the router right of A. Hence both the left and right routers send along all 4 directions N, S, E and W. Here, there arises a problem. When the router to the left of A sends the packet along East, the packet comes back to source A again which again scans its VCT table and resends the packet to its left and right routers again. Thus, the packet keeps going back and forth and multiplying until network congestion leads to saturation.

Hence without additional techniques to avoid these faults, there is no way to implement lookahead VCTM without maintaining the bitmap for each next hop router separately which leads to significant additional area overhead thus making optimization necessary.

The second multicast scheme under consideration is RPM. The way RPM implements multicasting is by recursively partitioning the network around the current router into eight partitions. A signal from each of these partitions specify if there is any destination for the packet that resides inside that partition. The signal is set to 1 if a destination resides inside the partition. Using the eight signals from the eight partitions, the RPM algorithm determines which output ports the packets must be forwarded to. All this is done at the RC stage. Hence, the RPM logic is integrated into the RC stage. The RPM logic consists of a partitioning logic (that generates the eight signals) and the routing decision making logic.

In case of RPM with lookahead routing, there might be many next hop routers due to the presence of multicast branches. This makes it necessary for the RC stage to determine the output ports for each of the next hop router. This can be done by performing the RC stage as many times as the number of branches of the packet. But this method will increase the router latency significantly and the purpose of lookahead routing will be defeated. On the other hand, the RC stage hardware can be replicated enough number of times so that each one of them can perform RC for one of the branches such that the RC stage can be completed in one cycle. Analysis needs to be done to know if this causes an area overhead. Generally, the RC stage is small enough that it is neglected in the area analysis of many area and power model simulators like Orion. But, the partitioning logic inside a RPM multicast RC stage scales with the size of the network. With futuristic routers consisting of hundreds of cores, the area overhead of the RC stage with lookahead routing might be an issue for large networks. Table I shows the area estimates for the partitioning logic with and without lookahead routing for different network sizes. Table I clearly shows that for large network sizes, the size of the RC stage with lookahead is much higher than that without lookahead routing.

Hence, area optimization for multicast lookahead routing is an advantage. In this thesis, RPM [22] is taken as an example and an area optimization mechanism is

Table I. A comparison of area of lookahead RC designs used in the RPM scheme for different network sizes.

Network Size ($N \times N$)	Area of basic RC stage in RPM scheme (μm^2)	Area of RC stage with lookahead routing used in RPM scheme (μm^2)
8×8	312.84	938.52
16×16	1073.16	3219.48
32×32	4114.4	12343.2

proposed so that lookahead routing can be implemented with minimal area.

B. Header Compression

Different compression schemes have been proposed for NOCs in the past. With increasing number of routers, it is difficult to manage network traffic. This is especially true in a broadcasting and multicasting environment. Compression schemes seem to be one of the ways to tackle this problem. Compression aids in reducing the number of bits transmitted over the network. This helps in reducing the communication latency since now the shorter packet can be transmitted faster over a link. With smaller packets, the number of times virtual channel allocation, switch arbitration, switch allocation, switch traversal, link switching, and other resource usage happens is lesser and hence the power consumption is lower.

The impact of compression on the memory system performance in a CMP environment has been studied in some research. It is shown that L2 cache compression can provide up to 26% performance speedup assuming a constant-delay interconnect model [26]. Many prior works have focused on cache compression. However, the work on cache and NIC (Network Interface Controller) compression by Das et al [26] provides insight into the ramifications of using an actual NOC model. This is especially important with NOCs starting to account for higher and higher percentage of the total power consumption. The work is based on findings that many benchmarks have significant number of continuous zeros and contain frequently repeated patterns of bits that can be compressed easily. The NIC is fit with a compressor/decompressor which are based on significance-based compression schemes. The decompressor is pipelined into 5 stages such that the decompression latency can be hidden up to 3 cycles under the network latency. Thus, effectively only a 2 cycle latency delay is incurred. Frequent-pattern compression is done in five cycles. The first cycle consists of decoding the length of each word using the prefix tags. In the next two cycles, the starting address of each word is calculated by adding each word's length cumulatively. The compressed data bits are decoded into uncompressed words by a parallel pattern decoder using the prefix bits. Another useful compression scheme is the USBR [27] (Unused Significant Bit Removal) proposed in the work by Simon Ogg et al. In this work, if in a block of bytes or words the most significant bits dont change, then those bits are removed and a prefix is attached to keep track of it. This scheme is proposed in serial links connected to parallel-to-serial converters whose latency can be used to hide the compression-decompression latency. This scheme can have static block sizes as well as dynamic block sizes. Blocks can be merged if that gives a better compression ratio. In static block sizes, it is difficult to know what is the best block size. Too big blocks are difficult to store while too small makes it worthless to compress because of the extra compression overhead. Dynamic block size solves this problem by having the flexibility of varying sizes. The dynamic block sizing algorithm can be found in [27].

Equation 3.1 (Courtesy [27]) has to be satisfied if compression has to be achieved.

$$(LENGTH \times LSBS_{CH}) + MSBS_{SM} + OVHD < (LENGTH \times BITWIDTH) \quad (3.1)$$

where $LENGTH$ is the length of the block, $LSBS_{CH}$ is the number of least

Table II. Addresses with constant values in SPECint95 benchmarks.

Program	Constant Addresses
099.go	78.2%
124.m88ksim	99.3%
126.gcc	61.8%
130.li	28.8%
134.perl	80.4%
147.vortex	79.9%
129.compress	3.2%
132.ijpeg	6.7%

significant bits that change, $MSBS_{SM}$ is the number of most significant bits that stay the same, $OVHD$ is the overhead in bits and $BITWIDTH$ is the bit width of the data words. The packet header is generally ignored and the payload is compressed. The compression scheme is used with a transition reduction scheme called SILENT. The amount of bit reductions was from 100% to 53% for fixed block and 100% to 57% for dynamic sized blocks with the fixed block size set at 64 bits.

One other compression scheme is Frequent Value Locality and Value centric data cache design as described in [21]. The main contribution of this work is to introduce the frequent value locality. Frequent value locality means that some values appear to occur and be accessed more frequently in memory. The work also claims that out of the eight SPECint95 benchmarks, six exhibit this property. Moreover, in these six benchmarks, 10 distinct values occupy nearly 50% of all memory locations. This phenomenon is exploited by the compression scheme. Table II shows a list of addresses that have constant values as tabulated in Table 4 in [21].The percentage of constant values clearly show why these benchmarks are such good candidates for compression.

Many other compression schemes exist. But these compression schemes have not entered the territory of compression inside routers. The compressor and decompressor are always outside the NOC in the NIC or the cache. Moreover, these compression schemes don't talk about header flit compression on the fly. In a multicasting scenario in NOCs, the header flits become typically huge for many multicast schemes like the RPM as the number of nodes in the network increases. For such schemes, it is a really good idea to compress the header flits and save a flit or two. For example, in the RPM multicast scheme, the header flit contains a bitmap of all the nodes in the network. If the network size is 16×16, the size of the bitmap in the header flit is 256 bits. If the link width is 128 bits and is the same as flit width, then the head flit takes 2 flits to be transmitted. Considering that most of the messages have an average size of 4 flits in the case of one-flit headers and 5 flits for 2-flit headers, saving a flit through compression can lead to a significant improvement in performance. Taking this into consideration, the thesis proposes a header flit compression algorithm and analyzes the performance improvements obtained.

C. Motivation from RPM Header

Let us consider a 4×4 2-D mesh network as shown in the Fig. 14. Let the source be router 9 and the destinations be routers 0, 1, 2, 3, 12 and 14. Since, there are a total of 16 nodes in the network, encoding the destinations as a bit string needs 16 bits to carry the destination list in the worst case. But there are some interesting observations unique to RPM that make RPM a good candidate for header compression. When source node 9 replicates the packet into two copies and forwards one northwards and one southwards, the north packet header has zero set for all the destinations to the south of the forwarding router. Thus, in one of the two packets in the least, more than half of the destination bits in the header flit will be zeros. This creates a continuous sequence of zeros. In our example, there are twelve continuous zeros in the North packet header and twelve continuous zeros in the South packet header also. In each replicated packet, the destinations opposite to its direction of travel will be set to zero in the header flit. Hence the chances of finding a continuous sequence of zeros like we found in our example in Fig. 14 is very high.

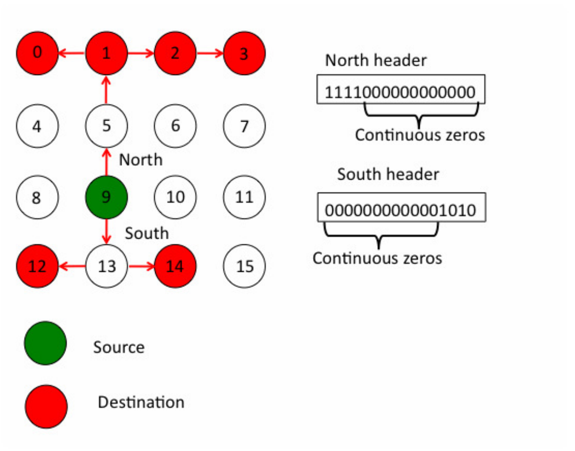


Fig. 14. Example header flit patterns in different directions. Continuous zeros are found frequently in the header.

One way of compressing these zeros is to encode the continuous sequence of zeros using a prefix to tell how many zeros were compressed and the starting position of the zero sequence. The rest of the header can be transmitted in the original unchanged format. Many multicast schemes have compression in-built in themselves. For example, the VCTM multicasting algorithm transmits just a VCT number and the routers use this Id to access their VCT tables and find the output ports to send. In our scheme, we don't even need to decompress the compressed zeros sequence since they will not be used anymore. They will not undergo any change for the rest of the packet's life. Therefore all the necessary information can be obtained without decompression. If we use lookahead routing, we can overlap the compression stage with the VA and SA stages. Therefore, we can assume that the compression stage is not in the critical path of the router pipeline and will not affect router latency.

CHAPTER IV

IMPLEMENTATION

A. RPM Area Optimization Algorithm

The RPM Area Optimization technique has been implemented over the RPM Multicast routing algorithm. As we saw earlier, as the number of cores increases, the RPM partitioning logic area overhead also increases. For large networks, the area of the partitioning logic cannot be passed over. With lookahead multicasting in place, three RC stages are needed for the RPM to calculate the output ports for up to three next hop routers. This might increase the area overhead of the RPM partitioning logic up to 3 times. This can be avoided if we can integrate the three stages into one RC stage. But, there is a problem with the integration of the RC stages together. The partitions of the network for each next hop router are different from each other. To overcome this problem, we use the area optimization technique. This technique exploits the overlap among the partitions of different next hop routers. With the area optimization technique, it is possible to combine the three RC stages into a single RC stage with only minimal overhead.

Let us consider an example 8x8 network implementing lookahead routing as shown in Fig. 15. The black rectangular blocks are the partitions for the next hop router colored black and the red rectangular blocks are partitions for the red colored next hop router. The overlap among the partitions is pretty good which can be exploited for the RC stage integration. The area optimization technique does precisely this by implementing another way of partitioning the network. Instead of eight partitions that were used in the original method, now we use twenty four partitions as shown in Fig. 16.

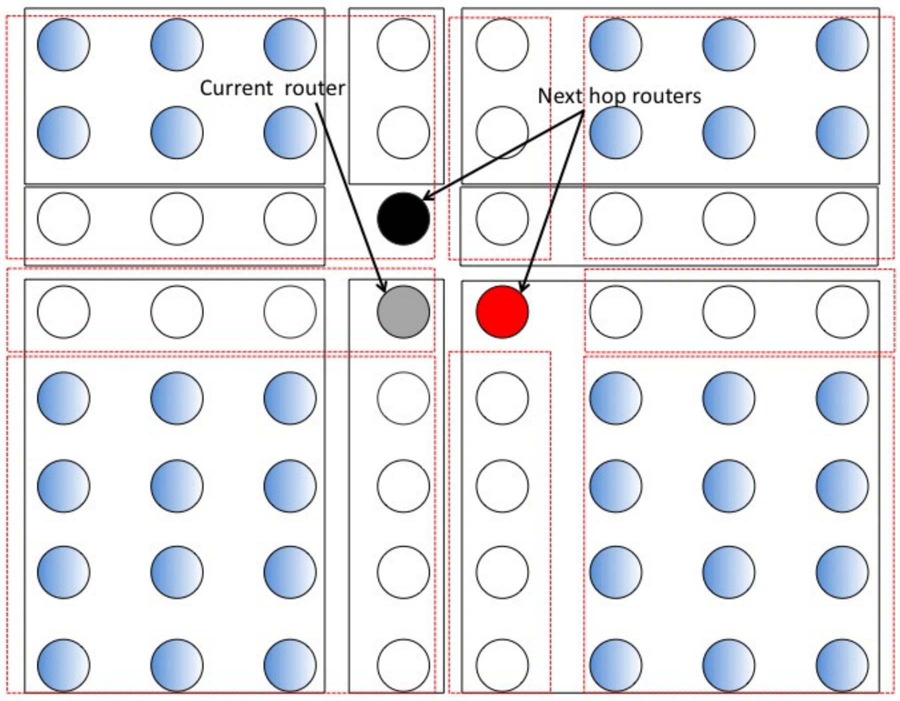


Fig. 15. Partition diagram for two out of four next hop routers. They are shown in red and black and the current router is shown in gray. Overlap among partitions is shown in blue.

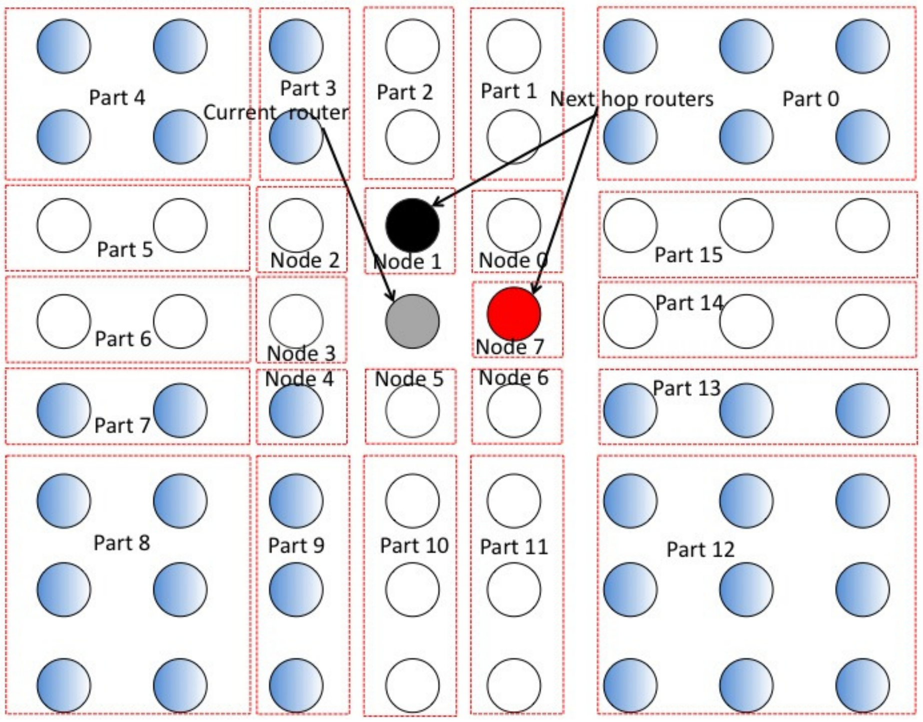


Fig. 16. The new partitioning scheme proposed for area optimization in lookahead routing.

The 8 partitions node 0-7 are single nodes that immediately surround the current router. The partitions part 0-15 are shown in Fig. 16. Part 1 is a single column matrix above node 0. Part 2 and part 3 are single column matrices above node 1 and node 2. Similarly, part 9, part 10 and part 11 are single column matrices below node 4, node 5, and node 6 respectively. Likewise, part 5, 6, 7 are single row matrix partitions to the left of node 2, node 3, and node 4 and part 13, 14, 15 are single row matrix partitions to the right of node 6, node 7 and node 0 respectively. These partitions have to be made only once and using these, the partitions for different next-hop routers of the current router can be computed with no or minimal overhead. The part signals are computed for the 24 partitions by OR-ing the destination bits from the header bit vector like it is done in the original RPM scheme. Let us call the signals from part 0-15 as P0-15 and the signals from nodes 0-7 as N0-7. These signals are computed just once and not thrice. To understand how to compute the 8 part signals as in the original scheme, let us try to find the part 0 signal (part 0 of the original eight partition scheme) for the next hop router in the North (colored black in Fig. 16) using the P0-15 and N0-7 signals. For the sake of clarity, let us temporarily call the new partitions as segments and retain the term "part" for the old 8 partition scheme. Part 0 is the northeast partition that comprises of the segments 0 and 1. Therefore, we can get the part 0 signal by OR-ing P0 and P1. The part 7 for the same next hop router is signal N0 OR-ed with P15 because it comprises of segment 15 and node 0. Similarly, all the eight part signals for all the routers can be computed. A list of how to calculate the different part signals for each of the four next hop routers is shown in Table III. The current router need not be included in the computation because the multicast packet has already reached the current router and therefore its bit is going to set to 0 in the header flit's destination list. It can be noted that the number of OR-gates used in the table is a constant.

Table III. Computation of part signals for next-hop routers.

Partition Signal (0-7)	Direction of next hop router (N, S, E, W)	Computation using P0-15 and N0-7 and OR gates
part 0	N	P0 OR P1
	S	N7 OR N0 OR P0 OR P1 OR P14 OR P15
	E	P0 OR P15
	W	N0 OR N1 OR P0 OR P1 OR P2 OR P15
part 1	N	P2
	S	P2 OR N1
	E	N0 OR P1
	W	N2 OR P3
part 2	N	P3 OR P4
	S	N3 OR N2 OR P3 OR P4 OR P5 OR P6
	E	N1 OR N2 OR P2 OR P3 OR P4 OR P5
	W	P4 OR P5
part 3	N	N2 OR P5
	S	N4 OR P7
	E	N3 OR P6
	W	P6
part 4	N	N3 OR N4 OR P6 OR P7 OR P8 OR P9
	S	P8 OR P9
	E	N4 OR N5 OR P7 OR P8 OR P9 OR P10
	W	P7 OR P8
part 5	N	N5 OR P10
	S	P10
	E	N6 OR P11
	W	N4 OR P9
part 6	N	N6 OR N7 OR P11 OR P12 OR P13 OR P14
	S	P11 OR P12
	E	P12 OR P13
	W	N5 OR N6 OR P10 OR P11 OR P12 OR P13
part 7	N	N0 OR P15
	S	N6 OR P13
	E	P14
	W	N7 OR P14

Let us look at the total number of OR gates used in a partitioning logic without lookahead and with lookahead. The number of OR gates needed to OR n destinations is n-1. Let us consider a $n \times n$ network . Fig. 17 shows the dimensions (in terms of rows and columns of nodes) of the partitions for a 8-partition scheme with no lookahead multicast routing. Considering part 2 for instance, the number of OR gates is pq -1 . Similarly, calculating and adding the number of OR gates in all the partitions, we get the total number of OR gates for a 8-partition RPM partitioning logic without lookahead to be $n^2 - 9$. Fig. 18 similarly shows the dimensions for a RPM scheme with 24 partitions used in lookahead multicast routing. The number of OR gates can be calculated to be $n^2 - 25$. But, the number of gates from Table III has to be added which is 60. Therefore, the total number of OR gates for the new RPM design is $n^2 + 35$ which is only around 45 OR gates more than the 8-partition scheme without lookahead. Comparing with the overhead that would occur if we replicated the RC stage three times, this is negligible, especially as the network size grows, because it is a constant.

B. Header Compression Scheme

Bearing in mind RPM's recursive partitioning feature at each intermediate node, we propose a new compression scheme which can efficiently reduce the overhead of a packet header.

The original packet header format of RPM is shown in Fig. 19(a), while a new format is shown in Fig. 19(b). The original header contains destinations in serial order from 0 to n-1. The new header format is implemented to exploit the features provided by RPM partition logic. It includes a compression bit to indicate if the header is compressed or not. The compression bit is followed by three relevant part

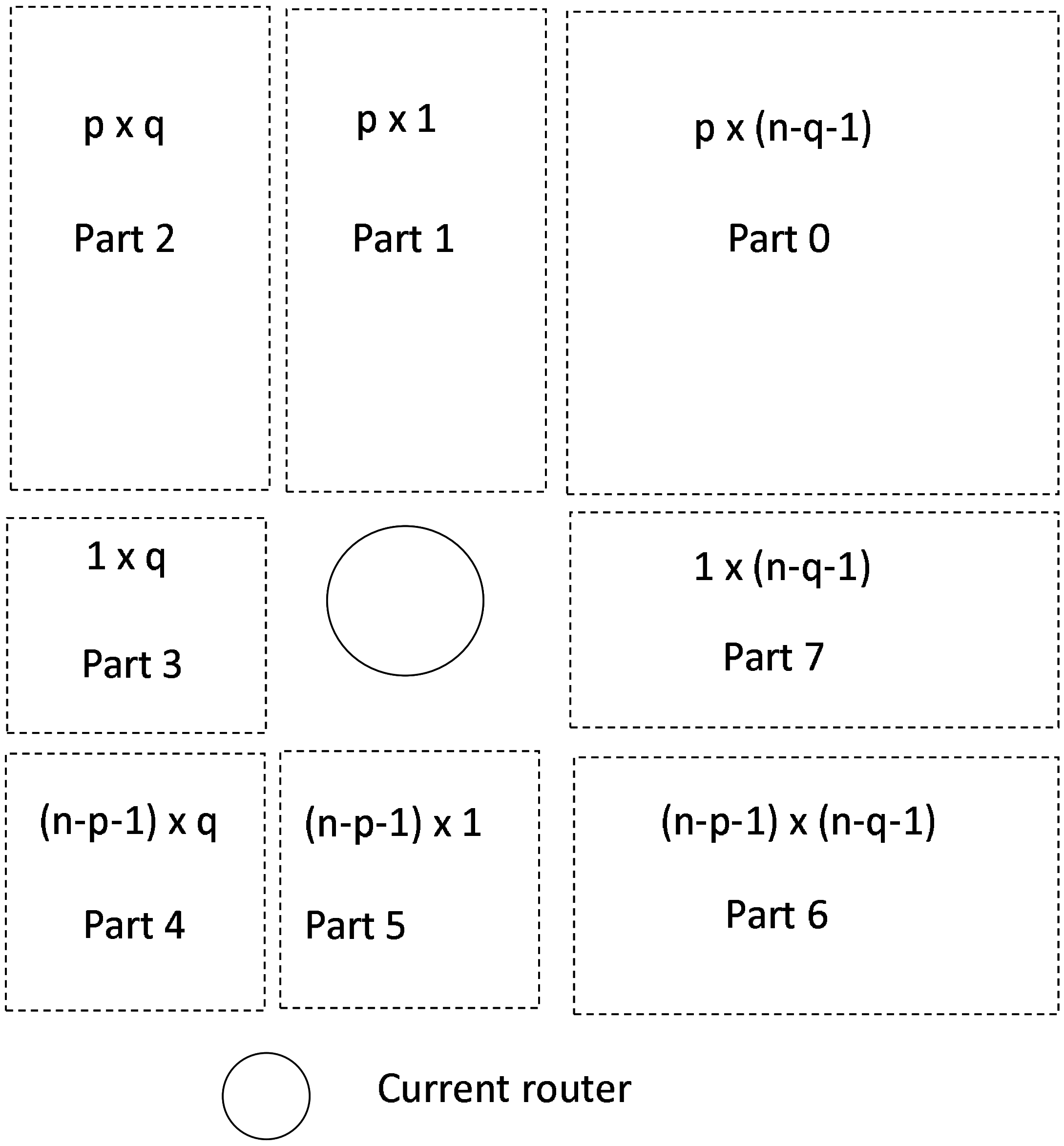


Fig. 17. Gate count for old partition scheme.

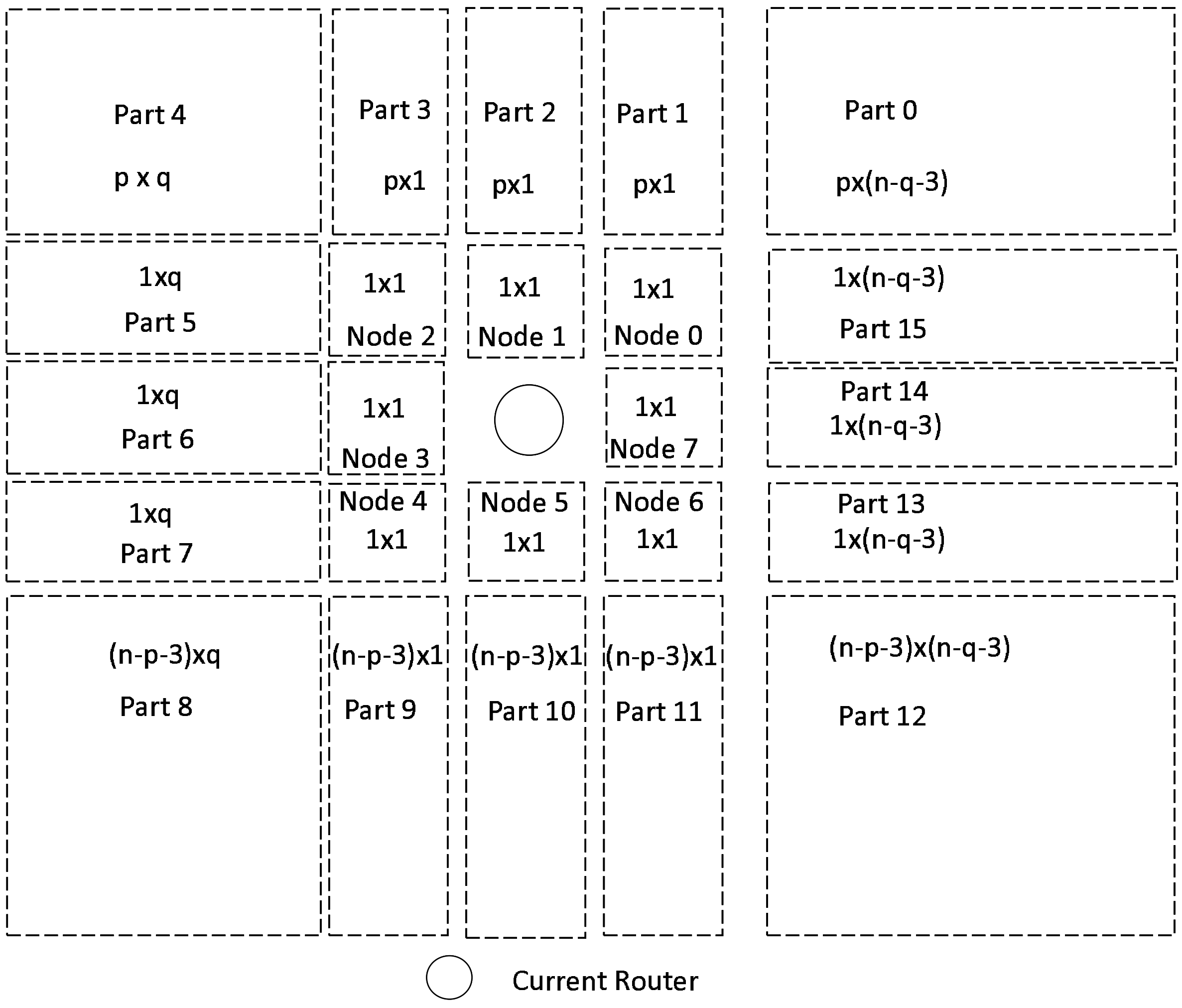


Fig. 18. Gate count for new partition scheme.

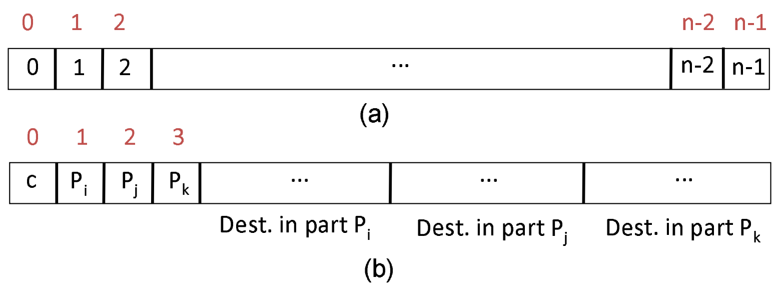


Fig. 19. RPM header format and proposed header compression format.

signal bits P$_i$, P$_j$ and P$_k$ which are obtained from the RPM partition logic. Other bits following these part signal bits represent the destinations that belong to the corresponding parts i, j and k. Parts i, j and k are selected based on the direction in which the packet will be sent. For instance, if the packet is sent eastward, the parts i, j and k are respectively 0, 6 and 7 of the current router. Destinations in the other parts will not receive this east direction packet. Hence the bits corresponding to these destinations are always set to 0 in the header. Therefore, the bits in the header for these destinations are removed as depicted in Fig. 19(b). According to the policy of avoiding redundant packet replications in RPM, each direction can at most have destinations in three parts, which means using three bits for parts information in the header is always enough. The destination bits for the three parts follow row-wise ordering. The part signals P$_i$, P$_j$ and P$_k$ are set to 1 if at least one destination exists in their parts respectively. Any part signal being zero indicates that no destination exists in that part, which means that all the destination bits in that part are set to zero.

An example of this compressed packet header in a (4×4) network is shown in Fig. 20. It shows the current router and the direction in which the packet is traveling. The original packet header and new packet header formats are illustrated in Fig. 21(a) and (b). 16 routers require 16 bits in the original header format. The size of the new

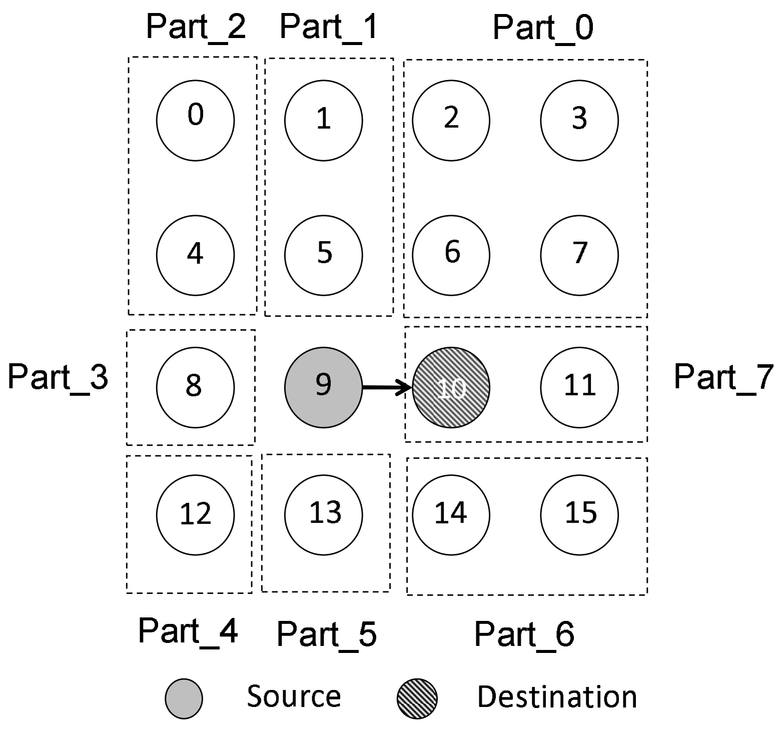


Fig. 20. In a (4×4) network, node 9 sends a multicast packet and node 10 is the only destination. The whole network is divided into eight parts according to the position of node 9.

header varies depending on the position of the current router and the destination list. In this example, Router 9 sends a packet to Router 10. Parts i, j, k are 0, 6 and 7 respectively. The destination bits are Routers 2, 3, 6, 7 for part 0, Routers 14, 15 for part 6 and Routers 10, 11 for part 7. Since the destination is only 10, P_0 and P_6 are 0 while P_7 is 1. The destination bits for P_0 (2, 3, 6, 7) and P_6 (14, 15) are omitted, because the part bit P_0 set to 0 already carries the information that the destination bits for routers 2, 3, 6 and 7 are zeros, and similarly for P_6. The final header after compression is shown in Fig. 21(c). The first bit for compression is set to 1 which means that this header is compressed. The second, third and fourth bits are the part signals for parts 0, 6 and 7 which are 0, 0 and 1 respectively. The destination bits for part 7 whose part signal is 1 are set to 1 and 0 for Router 10 and 11 respectively. It

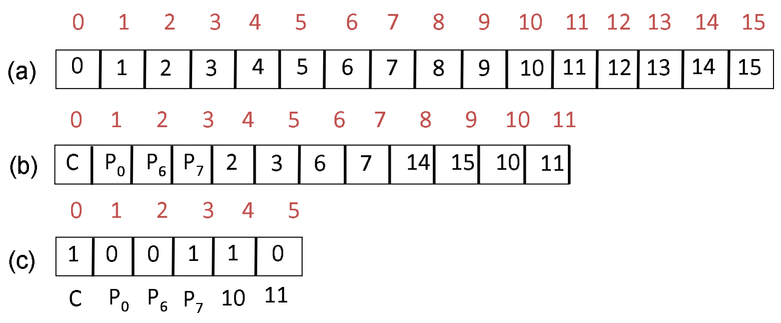


Fig. 21. Comparison of different headers. (a) RPM format, (b) Proposed header co mpression format and (c) the compressed header for the example in Fig. 20.

can be observed that the number of bits were reduced from 16 to just 6 in this case. The 6 bits include a compression overhead of 4 bits and just 2 bits for sending the actual destination bits. The compression overhead of 4 bits is a constant and will not increase with the size of the network. Furthermore, the partition logic can be used to implement the compression eliminating the need for a new compression logic.

Since partitioning in RPM is based on the position of the current router, routers at different positions will decode the same header in different ways. This also depends upon the traversal direction of packets. However, the header decoding for a particular router in a dedicated direction is fixed and can be implemented by synthesized hardware logic for each router. Also, the size of a compressed header can vary from packet to packet depending on whether each of the part signals is set to 0 or 1. For instance, in Fig. 20, if Routers 6 and 7 are packet destinations, then part 0 would be set to 1 by the RPM partition logic and hence the destination bits 2, 3, 6 and 7 would be included in the header. Therefore, the location of the bits for destination routers 10 and 11 in the header should be changed. In this way, decoding logic of the packet header needs multiplexing among the different possible locations for each destination bit and using P_i, P_j and P_k as the selection bits. Fig. 22 shows the hardware of the decompression stage in Router 10 in a (4×4) network for a packet sent from Router

9. The decompression logic needs to decode the destination bits from the header and forward it to the RPM logic for route computation. The destination bits for Routers 2, 3, 6 and 7 are located at the bit number 4, 5, 6 and 7 if the bit P_0 is 1, otherwise they will not be present in the header. These four destination bits can be extracted from the header with four multiplexers. The destination bits 14 and 15 can be located (if P_6 is set to 1) at bit numbers 8, 9 or 4, 5 depending on whether P_0 is set to 1 or 0 respectively. Hence, the extraction of these destination bits need an extra stage of multiplexing. Similarly, the destination bits of Routers 10 and 11 can have four different locations and can be multiplexed using the combination of P_0 and P_6 as the selection signals. In a similar way, the logic can be customized for every input port of each rout er.

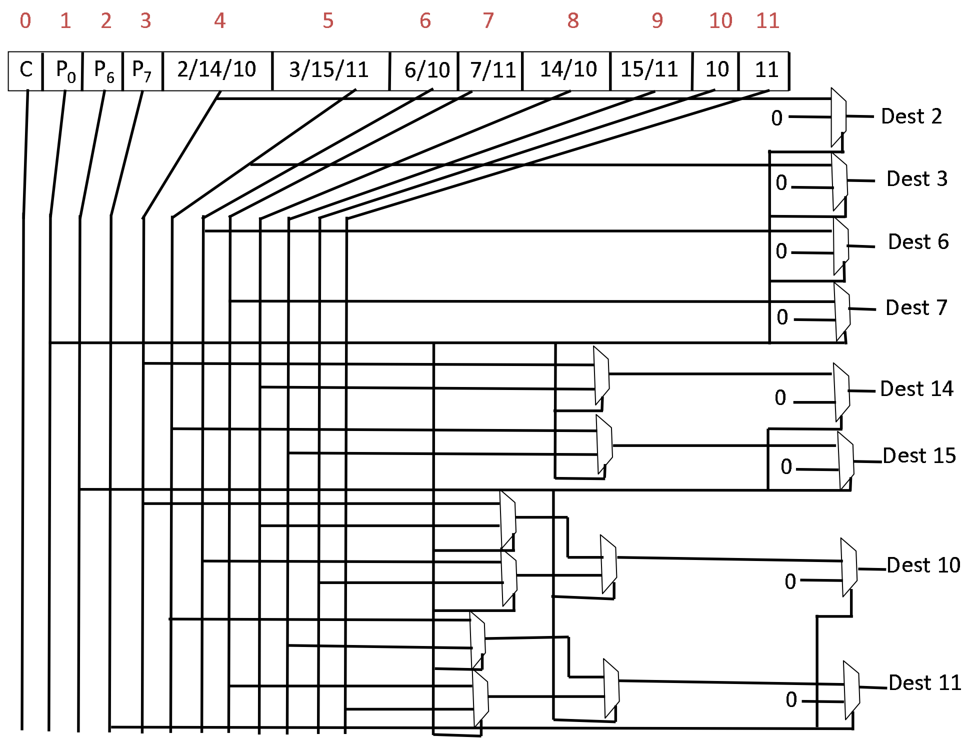


Fig. 22. An implementation of decode component in router 10 for packets from router 9.

In some special cases, one packet header needs to carry a huge destination list. For example, say we have 1K nodes and only a 128-bit header. Even with compression, it is hard to put the whole destination list into one flit. Then at the source node, we can use more than one flit to carry the destination list. However, as the packet proceeds to the next hop, according to RPM, it might be replicated to different directions. During the replication process, the original destination list will be split into several parts to make a new header for each replica. In this way, the header of each replica will get rid of many destinations, and hence, will have many zeros which makes it easier to be compressed. This procedure will recursively occur along the whole path until every destination receives a copy of the original packet. Detailed analysis of our compression scheme will be carried out in Chapter V.

CHAPTER V

EXPERIMENTAL EVALUATION

We evaluate the area overhead of our design, especially for the RC logic. We then analyze the effectiveness of our compression scheme. We also evaluate the performance of our multicast router design with different synthetic multicast workloads.

A. Methodology

We use a cycle-accurate network simulator that models all router pipeline delays and wire latencies. We use Orion 2.0 [28] for area estimation. Orion 2.0 simulator uses a recent model [29] and estimates the area of transistors and gates using the analysis in [30]. The area depends on the technology-level and process-level input parameters. Orion 2.0 simulator provides value estimates for inverters and 2-input AND and NOR gates. The simulator also adds an additional 10% to the total area to account for global white space. We model a link as 128 parallel wires, which takes advantage of abundant metal resources provided by future multi-layer interconnects. On top of Uniform Random (UR), Bit Complement (BC) and Transpose (TP) unicast packets, our synthetic workloads have multicast packets. For multicast packets, the destination numbers and positions are uniformly distributed, while unicast packets' destinations are determined by three patterns (UR, BC, and TP). We also control the percentage of multicast packets. Table IV summarizes the simulated configurations.

B. Compression Analysis

In this part, the efficiency of the proposed header compression scheme is evaluated with different network sizes. Here we define the "header" as the set of bits used to

Table IV. Network configuration.

Characteristic	Configuration
Topology	8×8 Mesh or 16×16 Mesh
Routing	RPM
Virtual Channels/Port	4
Virtual Channel Depth (flits)	4
Packet Length (flits)	4 or 5
Traffic Pattern	UR, BC and TP
Multicast Packet Portion	10%
Multicast Destination Number	2-16 (uniformly distributed)
Simulation Warm-up Cycles	10,000
Total Simulation Cycles	20,000

indicate the multicast destinations. The number of destinations is randomly selected and varies from 1, which implies a unicast packet, to the total number of nodes which refers to broadcasting. The original number of bits in the header is equal to the number of nodes in the network. For instance, the number for a 4×4 network is 16 and for a 16×16 network is 256. Without compression, the number remains fixed as the packet is replicated and forwarded along the network path. Hence the average size of the uncompressed header is n^2 for a $n \times n$ network. On the other hand, the size of a compressed header can be different depending on the extent of compression. Even at the source node, different multicast packets can have different header sizes. Furthermore, as one packet traverses, the destination list may be split into many parts, which means the replicated packet header will have a better chance to get compressed. In our simulation, we consider this effect.

As clearly illustrated in Fig. 23, the header size without any compression is bigger than the other two. Compression at the source node yields around 45% reduction in the header size for all the network sizes except the 4×4 network which yields approximately 25% reduction. The reduction from compression is predominantly due to the elimination of the bits for destinations in certain parts which are not included in some direction. The 4×4 network is an exception because the 4-bit compression overhead becomes significant when the network is small. This overhead is negligible in large networks. Looking at Fig. 23, compression at all routers yields more significant header size reduction. Comparing the results between the header size at the source and the average header size at all intermediate routers, it can be inferred that as the packet traverses the downstream routers, there is more and more opportunity for compression due to the characteristics of RPM. As a packet is forwarded downstream, RPM divides the destinations among the replicated packets in different directions, leading to the removal of certain other destination bits from the packet header. Thus the number of destinations in each packet decreases. Therefore, a lot of bits in the header tend to be zeros which creates a high chance for compression. The compression rate ranges from 78% for a 8×8 network to 96% for a 32×32 network.

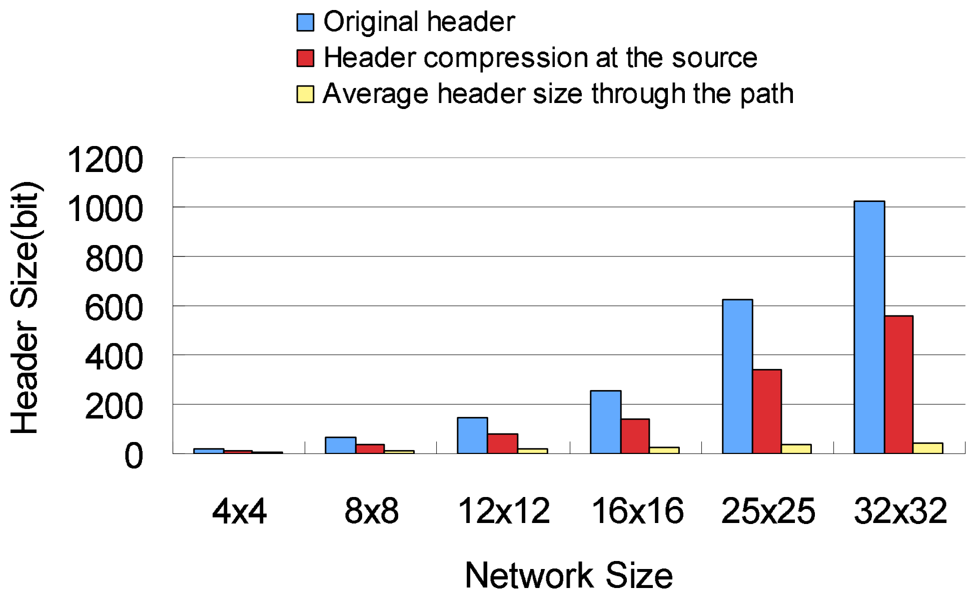


Fig. 23. Comparison of original header and the header after compression for different network sizes.

Fig. 24 shows another interesting result of our compression scheme. In this experiment, we fix the network size as 16×16 and keep changing the number of destinations. The position of each destination is randomly selected. As Fig. 24 illustrates, even with a large number of destinations, the average size of the header does not increase. One explanation can be from the number of replications. We can see that the larger the number of destinations, the higher the number of replications. Each replication splits the header into more parts with less destinations in each of them thus creating a better opportunity for compression.

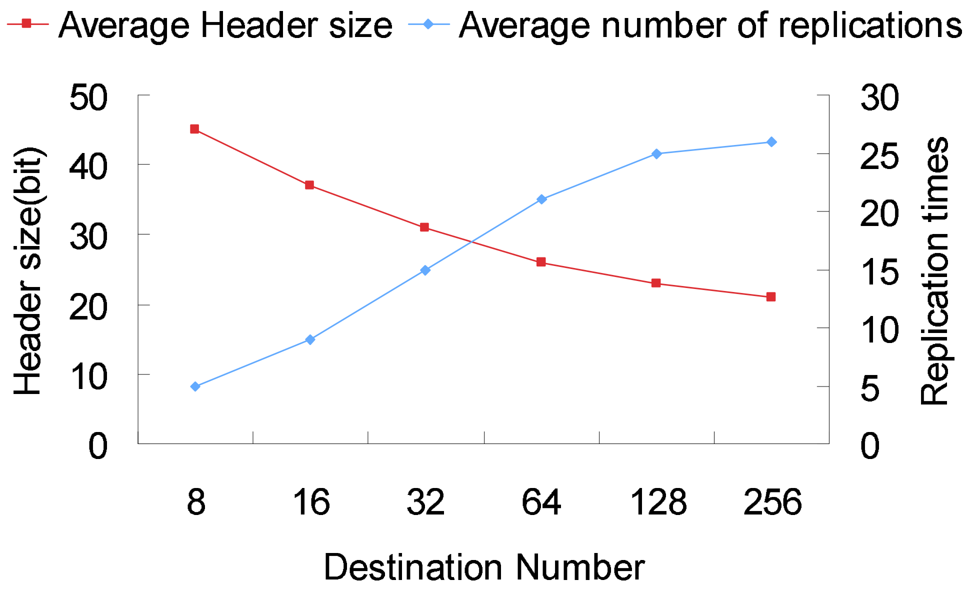


Fig. 24. Header compression with different number of destinations in a 16×16 network.

C. Area and Power Analysis

In our area analysis, we focus on the RC component since it scales with the number of routers and increases dramatically in size with lookahead routing. The area of the RC logic depends on the technology-level and process-input parameters. The Orion-2.0 simulator provides value estimates for inverters and 2-input AND and NOR gates. The area estimates for 2-input AND and NOR gates are shown in Table V.

We evaluate dynamic and static power consumption for RC component with 50% switching activity and 1V supply voltage in 65 nm technology. Table VI shows the area of the partitioning logic without lookahead routing for different network sizes.

Table V. Area of INVERTER, NOR and AND gates from Orion 2.0.

Process Technology (nm)	SCALE_T	Area of Inverter gate (μm^2)	Area of AND gate (μm^2)	Area of NOR gate (μm^2)
90	1	2.82	4.23	4.23
65	1	1.44	2.52	2.52
45	0.7343760	1.0575	1.8506	1.8506
32	0.5162063	0.7434	1.3003	1.3003

Table VI. Number of gates for basic multicast RC.

Network Size	NOR gates	INV gates	AND gates
4×4	23	31	8
8×8	71	79	8
16×16	263	271	8
32×32	1031	1039	8

From the table, we can see that the area of RC logic increases linearly with network size. The RC logic is implemented using NOR, Inverter and AND gates. The number of Inverter and NOR gates is almost the same as the total number of routers in the network. The RPM direction calculation logic adds a constant to the total number of gates.

With lookahead routing enabled, if the area optimization is not used, the RC logic needs to be replicated three times and hence the area triples. However, in our new RC design, we divide the network more finely into 24 parts instead of 8. We can generate the original 8 part signals for each next-hop router from the same 24 parts and achieve logic reuse. The area comparison between the two designs is given in Table VII. The table shows clearly area reduction with our RC design. The area reduction is more prominent as the size of the mesh increases. Fig. 25 illustrates the power consumption of the RC component. Since there are less logic gates in the optimized lookahead RC design, both static and dynamic power are saved.

Fig. 26 shows the comparison of power consumption per packet between headers with no compression and with compression. In this part, we evaluate the power for Uniform Random Traffic pattern in a 16×16 mesh network. The power value is recorded before the network is saturated. The power of the router with header

Table VII. Area of different lookahead RC designs.

Network Size	Area of basic RC (μm^2)	Area of basic RC with lookahead routing (μm^2)	Area of RC with lookahead routing and optimization (μm^2)
4×4	122.76	368.28	249.48
8×8	312.84	938.52	439.56
16×16	1073.16	3219.48	1199.88
32×32	4114.4	12343.2	4241.12

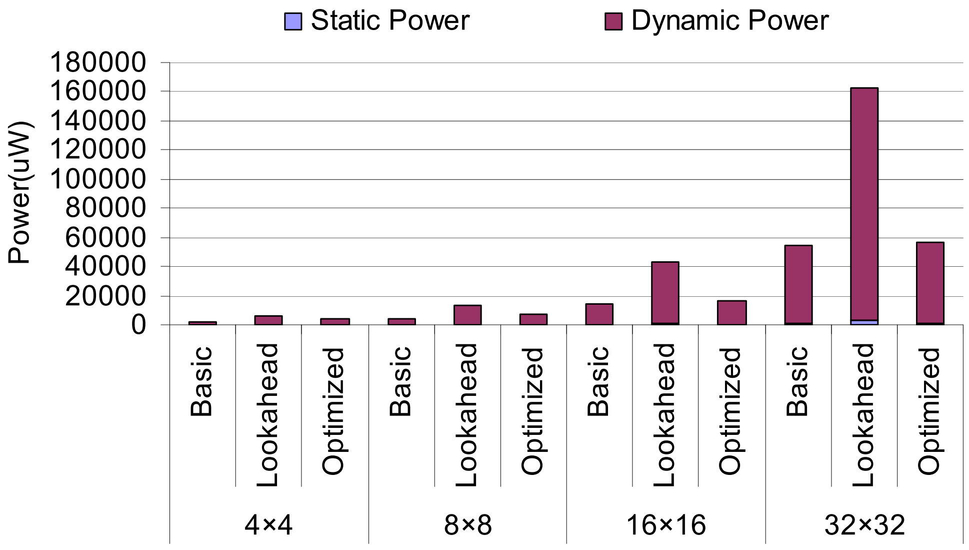


Fig. 25. RC power analysis.

compression is normalized to that of the router without header compression. We can see that header compression saves router power by 20% since it can reduce the packet header size.

D. Performance

Fig. 27 summarizes the performance results of lookahead routing in an 8×8 network for the three synthetic traffic patterns. In an 8×8 network, there are 64 nodes. Considering the link width is 128 bits, even without header compression, the entire destination list can be fit into one header flit. Therefore, in an 8×8 network, lookahead routing is independent of header compression. The simulation results are consistent

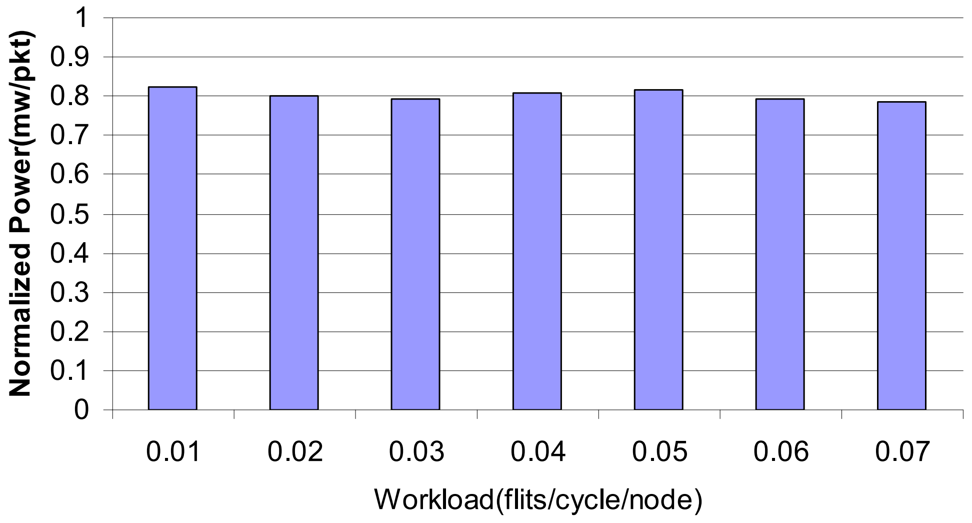
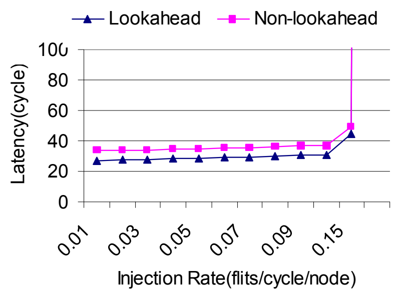


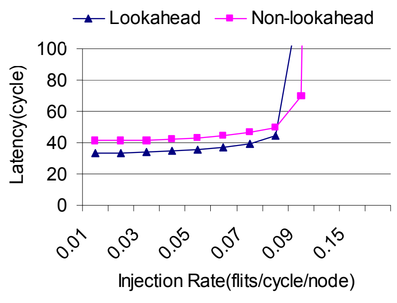
Fig. 26. Power analysis of header compression.

with our expectations. Lookahead routing removes one stage from the router pipeline, improving the packet latency but having little effect on network throughput.

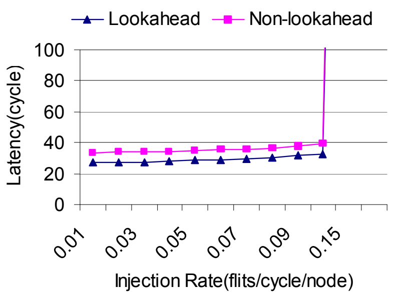
To study the effect of header compression, we choose a 16×16 mesh network where there are 256 nodes. Since the flit size is 128 bits (normally the flit size is the same as the link width), we need two flits to carry the whole destination list. With the same data payload, the packet size increases. Routing computation cannot be done in one cycle because it takes at least two cycles to get all the destination information. However, with header compression, most of the time the destination list can be fit into one flit. Even if the destination list cannot be compressed into one flit at the source node, as the packets hop along their paths, the original destination list might be split during multiple replications. Each replica only carries a part of the original set of destinations and therefore is easier to be compressed. Fig. 28 shows that the design with header compression and lookahead routing has the best performance for all the three traffic patterns. Without compression, lookahead routing is inefficient because the downstream router needs to wait more than one cycle to get the whole destination list, which negates the benefit of lookahead routing.



(a) UR(8 × 8)

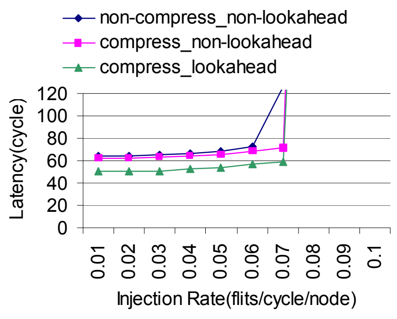


(b) BC(8 × 8)

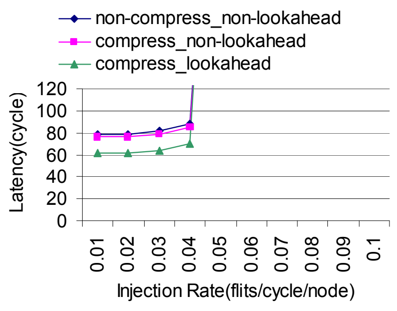


(c) TP(8 × 8)

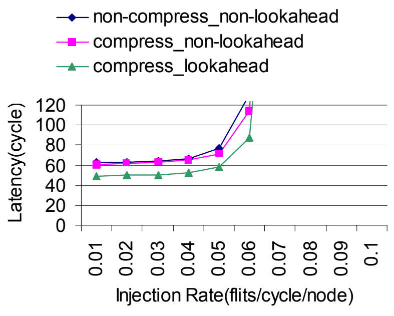
Fig. 27. Performance evaluation of multicast lookahead routing for three synthetic traffic patterns in a (8 × 8) mesh network (10% multicast traffic, average 8 destinations).



(a) UR(16 × 16)



(b) BC(16 × 16)



(c) TP(16 × 16)

Fig. 28. Packet latency for three synthetic traffic patterns in a (16 × 16) mesh network (10% multicast traffic, average 8 destinations). Lookahead routing has benefits only when header compression is applied.

CHAPTER VI

CONCLUSIONS

The prevalent use of NOCs in current CMP architectures indicates that it is important for NOCs to support multicast traffic. In this thesis, we explore the details of the multicast router design, especially the RC logic. We propose an efficient RC design to get rid of redundant gates while implementing lookahead routing. We also design a novel header compression scheme for multicast packets. With header compression, lookahead multicast routing in a large network becomes efficient. Simulation results show that with the new RC logic design with area optimization, providing lookahead routing in multicast routers only costs less than 20% area overhead over non-lookahead routing and this percentage keeps decreasing with larger network sizes. Compared with the basic lookahead routing design, our design save area by over 50%. With header compression and lookahead multicast routing, the network performance can be improved by over 22% in a large network.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, vol. 1, Morgan Kaufman Publishers, San Francisco, CA, 4th edition, 2007.

[2] J. P. Shen and M. H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, The McGraw-Hill Companies, Inc, New York, 1st edition, 2004.

[3] R. P. Colwell, R. P. Nix, J. J. O'Donnell, D. B. Papworth, and P. K. Rodman, "A VLIW architecture for a Trace Scheduling Compiler," *IEEE Transactions on Computers*, vol. 37, no. 8, pp. 967–979, August 1988.

[4] N. P. Jouppi and D. W. Wall, "Available instruction-level parallelism for superscalar and superpipelined machines," in *ASPLOS-III: Proc. 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, 1989, pp. 272–282, ACM.

[5] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *MICRO 24: Proc. 24th Annual International Symposium on Microarchitecture*, New York, 1991, pp. 51–61, ACM.

[6] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, "The agree predictor: a mechanism for reducing negative branch history interference," in *ISCA '97: Proc. 24th Annual International Symposium on Computer Architecture*, New York, 1997, pp. 284–291, ACM.

[7] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading:

maximizing on-chip parallelism," in *ISCA '95: Proc. 22nd Annual International Symposium on Computer Architecture*, New York, 1995, pp. 392–403, ACM.

[8] M. S. Schlansker and B. R. Rau, "EPIC: explicitly parallel instruction computing," *Computer*, vol. 33, no. 2, pp. 37–45, 2000.

[9] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," *SIGPLAN Not.*, vol. 31, no. 9, pp. 2–11, 1996.

[10] J. Liu, A. R. Mamidala, A. Vishnu, and D. K. Panda, "Evaluating infiniband performance with PCI express," *IEEE Micro*, vol. 25, no. 1, pp. 20–29, 2005.

[11] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway, "The AMD opteron processor for multiprocessor servers," *IEEE Micro*, vol. 23, no. 2, pp. 66–76, 2003.

[12] L. Bononi and N. Concer, "Simulation and analysis of network on chip architectures: ring, spidergon and 2d mesh," in *DATE '06: Proc. 9th Design, Automation and Test in Europe, 2006*, Munich, Germany, 6-10 March 2006, vol. 2, pp. 154–159.

[13] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[14] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[15] S. J. Eggers and R. H. Katz, "Evaluating the performance of four snooping cache coherency protocols," *SIGARCH Comput. Archit. News*, vol. 17, no. 3,

pp. 2–15, 1989.

[16] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam, "The stanford dash multiprocessor," *IEEE Computer*, vol. 25, no. 3, pp. 63–79, 1992.

[17] H. Gargia-Molina and A. Spauster, "Message ordering in a multicast environment," in *In Proc. 9th International Conference on Distributed Computing Systems*, Newport Beach, CA, 1989, pp. 354–361.

[18] J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Trans. Comput. Syst.*, vol. 2, no. 3, pp. 251–273, 1984.

[19] S. Ogg and B. Al-Hashimi, "Improved data compression for serial interconnected network on chip through unused significant bit removal," in *VLSID '06: Proc. 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design*, Washington, DC, 2006, pp. 525–529, IEEE Computer Society.

[20] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," in *ISCA '04: Proc. 31st Annual International Symposium on Computer Architecture*, Washington, DC, 2004, p. 212, IEEE Computer Society.

[21] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value-centric data cache design," *SIGPLAN Not.*, vol. 35, no. 11, pp. 150–159, 2000.

[22] L. Wang, Y. Jin, H. Kim, and E. J. Kim, "Recursive partitioning multicast: A bandwidth-efficient routing for networks-on-chip," in *NOCS '09: Proc. 3rd*

ACM/IEEE International Symposium on Networks-on-Chip. 2009, Washington, DC, 2009, pp. 64–73, IEEE Computer Society.

[23] N. E. Jerger, L.-S. Peh, and M. Lipasti, “Virtual circuit tree multicasting: A case for on-chip hardware multicast support,” in *ISCA ’08: Proc. 35th Annual International Symposium on Computer Architecture*, Washington, DC, 2008, pp. 229–240, IEEE Computer Society.

[24] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, “Efficient unicast and multicast support for cmps,” in *MICRO 41: Proc. 41st Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, 2008, pp. 364–375, IEEE Computer Society.

[25] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[26] R. Das, A. K. Mishra, C. Nicopoulos, Dongkook Park, V. Narayanan, R. Iyer, M. S. Yousif, and C. R. Das, “Performance and power optimization through data compression in network-on-chip architectures,” in *HPCA ’08: Proc. 14th IEEE International Symposium on High Performance Computer Architecture, 2008.*, Salt Lake City, UT, 16-20 Feb 2008, pp. 215 –225.

[27] S. Ogg and B. Al-Hashimi, “Improved data compression for serial interconnected network on chip through unused significant bit removal,” in *VLSID ’06: Proc. 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design*, Washington, DC, 2006, pp. 525–529, IEEE Computer Society.

[28] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, “ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration,” in *DATE*

'09: Proc. 12th Design, Automation and Test in Europe, 2009, Nice, France, 2009, pp. 423–428.

[29] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," Tech. Rep., HP Laboratories, 2008.

[30] H. Yoshida, K. De, and V. Boppana, "Accurate pre-layout estimation of standard cell characteristics," in *DAC '04: Proc. 41st Annual Design Automation Conference*, New York, 2004, pp. 208–211, ACM.

VITA

Poornachandran Kumar received his B.Tech degree in electronics and communication engineering from Indian Institute of Technology, Roorkee, India in 2005. He entered the Computer Engineering Graduate program at Texas A&M University in Fall 2007 and graduated with his M.S. in August 2010. His research interests include networks-on-chip architectures, cache coherence protocols, and chip multi-processors.

Poornachandran may be reached by email at kpc84uec@neo.tamu.edu or at the High Performance Computing Laboratory, H.R. Bright Building, Texas A&M University, College Station, 77843.