

# Microarchitecture of a High-Radix Router

John Kim, William J. Dally, Brian Towles<sup>1</sup>, Amit K. Gupta

Computer Systems Laboratory  
Stanford University, Stanford, CA 94305

<sup>1</sup>D.E. Shaw Research and Development  
New York, NY 10036

{jjk12, billd, btowles, agupta}@cva.stanford.edu

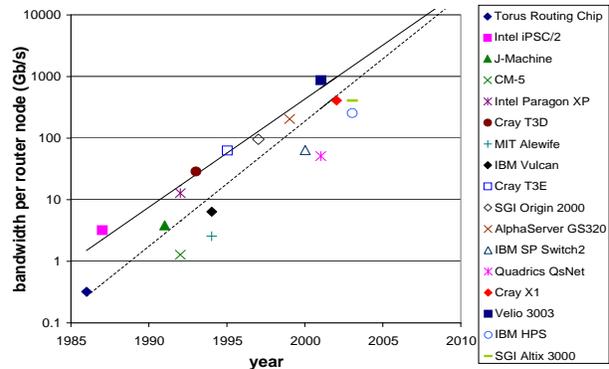
## Abstract

Evolving semiconductor and circuit technology has greatly increased the pin bandwidth available to a router chip. In the early 90s, routers were limited to 10Gb/s of pin bandwidth. Today 1Tb/s is feasible, and we expect 20Tb/s of I/O bandwidth by 2010. A high-radix router that provides many narrow ports is more effective in converting pin bandwidth to reduced latency and reduced cost than the alternative of building a router with a few wide ports. However, increasing the radix (or degree) of a router raises several challenges as internal switches and allocators scale as the square of the radix. This paper addresses these challenges by proposing and evaluating alternative microarchitectures for high radix routers. We show that the use of a hierarchical switch organization with per-virtual-channel buffers in each subswitch enables an area savings of 40% compared to a fully buffered crossbar and a throughput increase of 20-60% compared to a conventional crossbar implementation.

## 1 Introduction

Interconnection networks are widely used to connect processors and memories in multiprocessors, as switching fabrics for high-end routers and switches, and for connecting I/O devices. The interconnection network of a multiprocessor computer system is a critical factor in determining the performance of the machine. The latency and bandwidth of the network largely establish the remote memory access latency and bandwidth.

Advances in signaling technology have enabled new types of interconnection networks based on high-radix routers. The trend of increase in pin bandwidth to a router chip is shown in Figure 1 which plots the bandwidth per router node versus time. Over the past 20 years, there has been an order of magnitude increase in the off-chip bandwidth approximately every five years. This increase in bandwidth results from both the high-speed signaling technology [15, 21] as well as the increase in the number of signals available to a router chip. The advances in technology make it possible to build single chips with 1Tb/s of I/O bandwidth today [14], and by 2010, we expect to be able to put 20Tb/s of I/O band-



**Figure 1.** Router Scaling Relationship [2, 7, 9, 11, 13, 16, 20, 22, 26, 28, 30, 31]. The dotted line is a curve fit to all of the data. The solid line is a curve fit to the highest performance routers for a given time period.

width on a chip. This additional bandwidth is most effectively utilized and converted to lower cost and latency by increasing the radix or degree of the router.

Most implementations have taken advantage of increasing off-chip bandwidth by increasing the bandwidth per port rather than increasing the number of ports on the chip. However as off-chip bandwidth continues to increase, it is more efficient to exploit this bandwidth by increasing the number of ports — building high-radix routers with *thin* channels — than by making the ports wider — building low-radix routers with *fat* channels. We show that using a high radix reduces hop count and leads to a lower latency and a lower cost solution.

High-radix router design is qualitatively different from the design of low-radix high bandwidth routers. In this paper, we examine the most commonly used organization of a router, the input-queued crossbar, and the different microarchitectural issues that arise when we try to scale them to high-radix routers such as switch and virtual channel allocation. We present distributed allocator microarchitectures that can be efficiently scaled to high radix. Using intermediate buffering, different implementations of the crossbar for a high radix design are proposed and evaluated. We show that using a hierarchical switch organization leads to a 20-60%

increase in throughput compared to a conventional crossbar and provides an area savings of 40% compared to a fully buffered crossbar.

The rest of the paper is organized as follows. Section 2 provides background on the need for high-radix routers. Sections 3 to 6 incrementally develop the microarchitecture of a high-radix router, starting with a conventional router architecture and modifying it to overcome performance and area issues. Section 7 discusses additional simulation results. Section 8 discusses related work, and Section 9 presents conclusions.

## 2 The Need for High Radix Routers

Many of the earliest interconnection networks were designed using topologies such as butterflies or hypercubes, based on the simple observation that these topologies minimized hop count. The analysis of both Dally [8] and Agarwal [1] showed that under fixed packaging constraints, lower radix networks offered lower packet latency. The fundamental result of these authors still holds — technology and packaging constraints should drive topology design. What has changed in recent years are the topologies that these constraints lead us toward.

To understand how technology changes affect the optimal network radix, consider the latency ( $T$ ) of a packet traveling through a network. Under low loads, this latency is the sum of header latency and serialization latency. The header latency ( $T_h$ ) is the time for the beginning of a packet to traverse the network and is equal to the number of hops a packet takes times a per hop router delay ( $t_r$ ). Since packets are generally wider than the network channels, the body of the packet must be squeezed across the channel, incurring an additional serialization delay ( $T_s$ ). Thus, total delay can be written as

$$T = T_h + T_s = Ht_r + L/b \quad (1)$$

where  $H$  is the number of hops a packet travels,  $L$  is the length of a packet, and  $b$  is the bandwidth of the channels. For an  $N$  node network with radix  $k$  routers ( $k$  input channels and  $k$  output channels per router), the number of hops must be at least  $2\log_k N$ .<sup>1</sup> Also, if the total bandwidth of a router is  $B$ , that bandwidth is divided among the  $2k$  input and output channels and  $b = B/2k$ . Substituting this into the expression for latency from Equation (1)

$$T = 2t_r \log_k N + 2kL/B. \quad (2)$$

Then, setting  $dT/dk$  equal to zero and isolating  $k$  gives the optimal radix in terms of the network parameters,

$$k \log^2 k = \frac{Bt_r \log N}{L}. \quad (3)$$

<sup>1</sup>Uniform traffic is assumed and  $2\log_k N$  hops are required for a non-blocking network.

In this differentiation, we assume  $B$  and  $t_r$  are independent of the radix  $k$ . Since we are evaluating the optimal radix for a *given* bandwidth, we can assume  $B$  is independent of  $k$ . The  $t_r$  parameter is a function of  $k$  but has only a small impact on the total latency and has no impact on the optimal radix. Router delay  $t_r$  can be expressed as the number of pipeline stages ( $P$ ) times the cycle time ( $t_{cy}$ ). As radix increases,  $t_{cy}$  remains constant and  $P$  increases logarithmically. The number of pipeline stages  $P$  can be further broken down into a component that is independent of the radix ( $X$ ) and a component which is dependent on the radix ( $Y \log_2 k$ ). Thus router delay ( $t_r$ ) can be rewritten as

$$t_r = t_{cy}P = t_{cy}(X + Y \log_2 k).$$

If this relationship is substituted back into Equation (2) and differentiated, the dependency on radix  $k$  coming from the router delay disappears and does not change the optimal radix.<sup>2</sup> Intuitively, although a single router delay increases with a  $\log(k)$  dependence, the effect is offset in the network by the fact that the number of hop count decreases as  $1/\log(k)$  and as a result, the router delay does not effect the optimal radix.

In Equation (2), we ignore time of flight for packets to traverse the wires that make up the network channels. The time of flight does not depend on the radix( $k$ ) and thus has minimal impact on the optimal radix. Time of flight is  $D/v$  where  $D$  is the total physical distance traveled by a packet and  $v$  is the propagation velocity. As radix increases, the distance between two router nodes( $D_{hop}$ ) increases. However, the *total* distance traveled by a packet will be approximately equal since a lower-radix network requires more hops.

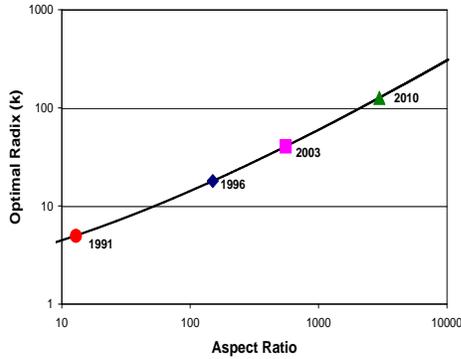
From Equation (3), we refer to the quantity  $A = \frac{Bt_r \log N}{L}$  as the *aspect ratio* of the router. This aspect ratio completely determines the router radix that minimizes network latency. A high aspect ratio implies a “tall, skinny” router (many, narrow channels) minimizes latency, while a low ratio implies a “short, fat” router (few, wide channels).

A plot of the minimum latency radix versus aspect ratio, from Equation (3) is shown in Figure 2. The points along the line show the aspect ratios from several years. These particular numbers are representative of large supercomputers with single-word network accesses<sup>3</sup>, but the general trend of the radix increasing significantly over time remains.

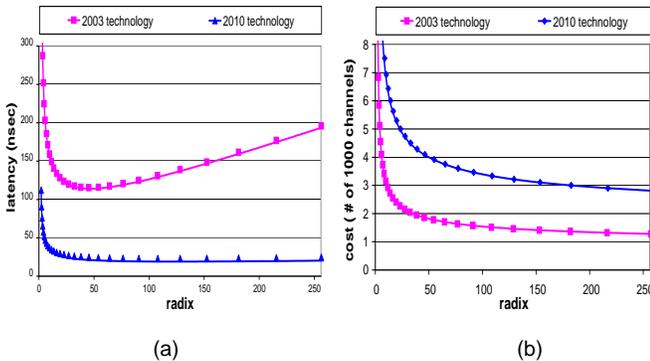
Figure 3(a) shows how latency varies with radix for 2003 and 2010 technologies. As radix is increased, latency first decreases as hop count, and hence  $T_h$ , is reduced. However, beyond a certain radix serialization latency begins to domi-

<sup>2</sup>If this detailed definition of  $t_r$  is used,  $t_r$  is replaced with  $Xt_{cy}$  in Equation (3).

<sup>3</sup>The 1991 data is from J-Machine [26] ( $B=3.84\text{Gb/s}$ ,  $t_r=62\text{ns}$ ,  $N=1024$ ,  $L=128\text{bits}$ ), the 1996 data is from the Cray T3E [30] ( $64\text{Gb/s}$ ,  $40\text{ns}$ ,  $2048$ ,  $128$ ), the 2003 data is from SGI Altix 3000 [31] ( $0.4\text{Tb/s}$ ,  $25\text{ns}$ ,  $1024$ ,  $128$ ) 2010 data is estimated( $20\text{Tb/s}$ ,  $5\text{ns}$ ,  $2048$ ,  $256$ ).



**Figure 2.** Relationship between optimal latency radix and router aspect ratio. The labeled points show the approximate aspect ratio for a given year's technology



**Figure 3.** (a) Latency and (b) cost of the network as the radix is increased for two different technologies.

nate the overall latency and latency increases. As bandwidth, and hence aspect ratio, is increased, the radix that gives minimum latency also increases. For 2003 technology (aspect ratio = 554) the optimum radix is 40 while for 2010 technology (aspect ratio = 2978) the optimum radix is 127.

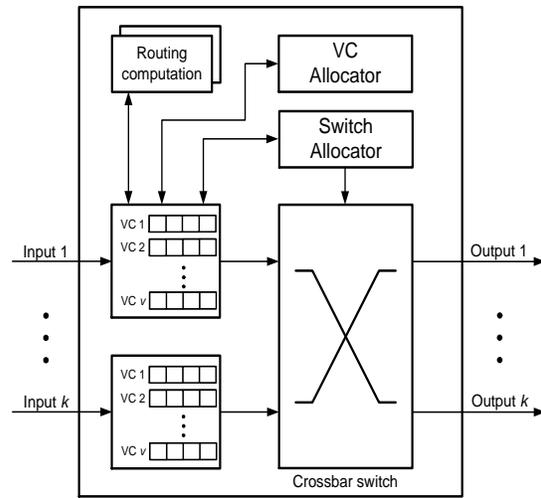
Increasing the radix of the routers in the network monotonically reduces the overall cost of a network. Network cost is largely due to router pins and connectors and hence is roughly proportional to total router bandwidth: the number of channels times their bandwidth. For a fixed network bisection bandwidth, this cost is proportional to hop count. Since increasing radix reduces hop count, higher radix networks have lower cost as shown in Figure 3(b).<sup>4</sup> Power dissipated by a network also decreases with increasing radix. Power is roughly proportional to the number of router nodes in the network. As radix increases, hop count decreases, and the number of router nodes decreases. The power of an individual router node is largely independent of

<sup>4</sup>2010 technology is shown to have higher cost than 2003 technology because the number of nodes is much greater.

the radix as long as the total router bandwidth is held constant. Router power is largely due to I/O circuits and switch bandwidth. The arbitration logic, which becomes more complex as radix increases, represents a negligible fraction of total power [33].

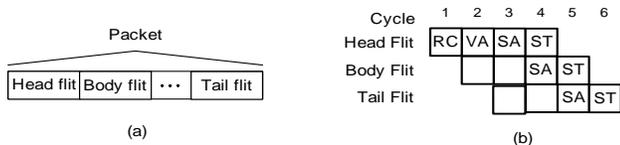
### 3 Baseline Router Architecture

The next four sections incrementally explore the micro-architectural space for a high-radix virtual-channel (VC) router. We start this section with a baseline router design, similar to that used for a low-radix router [24, 30]. We see that this design scales poorly to high radix due to the complexity of the allocators and the wiring needed to connect them to the input and output ports. In Section 4, we overcome these complexity issues by using distributed allocators and by simplifying virtual channel allocation. This results in a feasible router architecture, but poor performance due to head-of-line blocking. In Section 5, we show how to overcome the performance issues with this architecture by adding buffering at the switch crosspoints. This buffering eliminates head-of-line blocking by decoupling the input and output allocation of the switch. However, with even a modest number of virtual channels, the chip area required by these buffers is prohibitive. We overcome this area problem, while retaining good performance, by introducing a hierarchical switch organization in Section 6.



**Figure 4.** Baseline virtual channel router.

A block diagram of the baseline router architecture is shown in Figure 4. Arriving data is stored in the input buffers. These input buffers are typically separated into several parallel virtual channels that can be used to prevent deadlock, implement priority classes, and increase throughput by allowing blocked packets to be passed. The input



**Figure 5.** (a) Packets are broken into one or more flits (b) Example pipeline of flits through the baseline router.

buffers and other router resources are allocated in fixed-size units called *flits* and each packet is broken into one or more flits as shown in Figure 5(a).

The progression of a packet through this router can be separated into per-packet and per-flit steps. The per-packet actions are initiated as soon as the *header flit*, the first flit of a packet, arrives:

1. Route computation (RC) - based on information stored in the header, the output port of the packet is selected.
2. Virtual-channel allocation (VA) - a packet must gain exclusive access to a downstream virtual channel associated with the output port from route computation. Once these per-packet steps are completed, per-flit scheduling of the packet can begin.
3. Switch allocation (SA) - if there is a free buffer in its output virtual channel, a flit can vie for access to the crossbar.
4. Switch traversal (ST) - once a flit gains access to the crossbar, it can be transferred from its input buffers to its output and on to the downstream router.

These steps are repeated for each flit of the packet and upon the transmission of the *tail flit*, the final flit of a packet, the virtual channel is freed and is available for another packet. A simple pipeline diagram of this process is shown in Figure 5(b) for a three-flit packet assuming each step takes a single cycle.

## 4 Extending the baseline to high radix

As radix is increased, a centralized approach to allocation rapidly becomes infeasible — the wiring required, the die area, and the latency all increase to prohibitive levels. In this section, we introduce distributed structures for both switch and virtual channel allocation that scale well to high radices. In achieving this scalability, these structures compromise on performance.

### 4.1 Switch Allocation

We address the scalability of the switch allocator by using a distributed separable allocator design as shown in Figure 6. The allocation takes place in three stages: input arbitration,

local output arbitration, and global output arbitration. During the first stage all ready virtual channels in each input controller request access to the crossbar switch. The winning virtual channel in each input controller then forwards its request to the appropriate local output arbiter by driving the binary code for the requested output onto a per-input set of horizontal request lines.

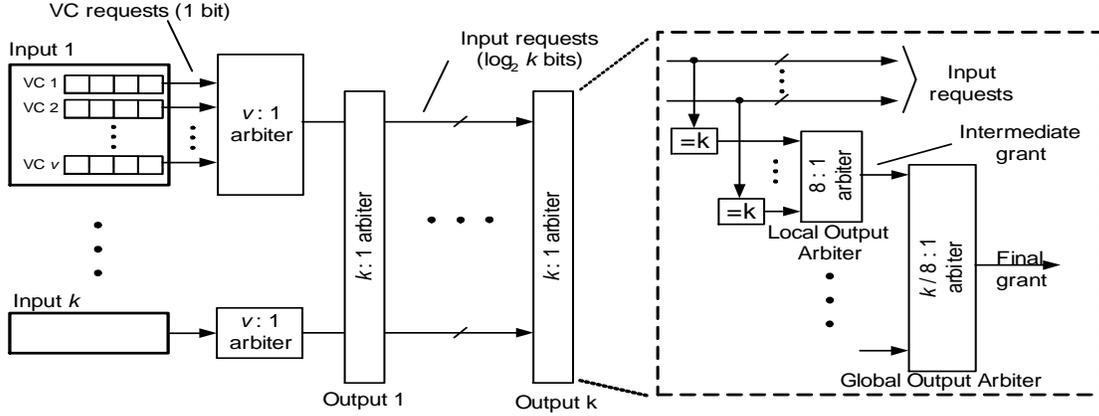
At each output arbiter, the input requests are decoded and, during stage two, each local output arbiter selects a request (if any) for its switch output from among a local group of  $m$  (in Figure 6,  $m = 8$ ) input requests and forwards this request to the global output arbiter. Finally, the global output arbiter selects a request (if any) from among the  $k/m$  local output arbiters to be granted access to its switch output. For very high-radix routers, the two-stage output arbiter can be extended to a larger number of stages.

At each stage of the distributed arbiter, the arbitration decision is made over a relatively small number of inputs (typically 16 or less) such that each stage can fit in a clock cycle. For the first two stages, the arbitration is also local - selecting among requests that are physically co-located. For the final stage, the distributed request signals are collected via global wiring to allow the actual arbitration to be performed locally. Once the winning requester for an output is known, a grant signal is propagated back through to the requesting input virtual channel. To ensure fairness, the arbiter at each stage maintains a priority pointer which rotates in a round-robin manner based on the requests.

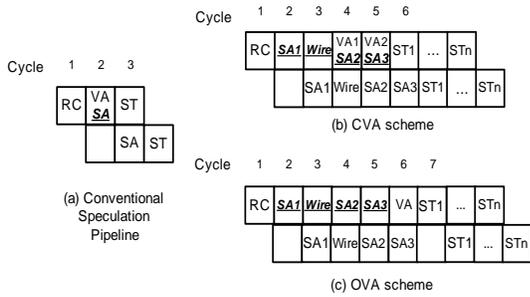
### 4.2 Virtual Channel Allocation

Virtual channel allocation (VA) poses an even more difficult problem than switch allocation because the number of resources to be allocated is multiplied by the number of virtual channels  $v$ . In contrast to switch allocation, where the availability of free downstream buffers is tracked with a credit count, with virtual channel allocation, the availability of downstream VCs is unknown. An ideal VC allocator would allow all input VCs to monitor the status of all output VCs they are waiting on. Such an allocator would be prohibitively expensive, with  $v^2k^2$  wiring complexity.

Building off the ideas developed for switch allocation, we introduce two scalable virtual channel allocator architectures. Crosspoint virtual channel allocation (CVA) maintains the state of the output virtual channels at each crosspoint and performs allocation at the crosspoints. In contrast, output virtual channel allocation (OVA) defers allocation to the output of the switch. Both CVA and OVA involve *speculation* where switch allocation proceeds before virtual channel allocation is complete to reduce latency. Simple virtual channel speculation was proposed in [27] where the switch allocation and the VC allocation occurs in parallel to reduce the critical path through the router (Figure 7(a)). With a



**Figure 6.** Scalable switch allocator architecture. The input arbiters are localized but the output arbiters are distributed across the router to limit wiring complexity. A detailed view of the output arbiter corresponding to output  $k$  is shown to the right.



**Figure 7.** Speculative pipeline with each packet assumed to be 2 flits. (a) speculation used on the pipeline shown in Figure 5(b) (b) high-radix routers with CVA (c) high-radix routers with OVA. The pipeline stages underlined show the stages that are speculative.

deeper pipeline in a high-radix router, VC allocation is resolved later in the pipeline. This leads to more aggressive speculation (Figure 7(b-c)).<sup>5</sup>

With CVA, VC allocation is performed at the crosspoints where the status of the output VCs is maintained. Input switch arbitration is done speculatively. Each cycle, each input controller drives a single request over a per-input set of horizontal virtual-channel-request lines to the local/global virtual output channel arbiter. Each such request includes both the requested output port and output virtual channel

The virtual channel allocator at each crosspoint includes a separate arbiter for each output virtual channel. Instead of

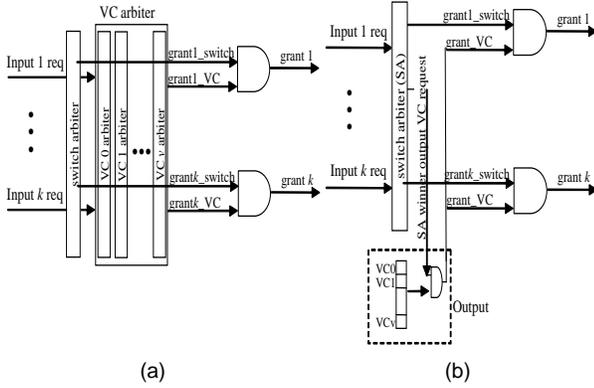
<sup>5</sup>Pipeline key: SAx: different stages of switch allocation, Wire: separate pipeline stage for the request from the input arbiters to travel to the output arbiters, STx: switch traversal, multiple cycles will be needed to traverse the switch

the  $k$  output arbiters used in the switch allocator (Figure 6), CVA uses a total of  $kv$  output virtual channel arbiters. Requests (if any) to each output virtual channel arbiter are decoded from the virtual channel request lines and each arbiter proceeds in the same local-global arbitration used in switch allocation.

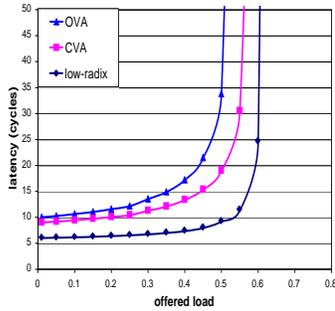
Using OVA reduces arbiter area at some expense in performance. In this scheme, the switch allocation proceeds through all three stages of arbitration and only when complete is the status of the output virtual channel checked. If the output VC is indeed free, it is allocated to the packet. As shown in Figure 7(c), OVA speculates deeper in the pipeline than CVA and reduces complexity by eliminating the per-VC arbiters at each crosspoint. However, OVA compromises performance by allowing only one VC per output to be requested per allocation. A block diagram of the different VA architectures is shown in Figure 8 and illustrates the control logic needed for the two schemes. They are compared and evaluated in the next section.

### 4.3 Performance

We use cycle accurate simulations to evaluate the performance of the scalable switch and virtual channel allocators. We simulate a radix-64 router using virtual-channel flow control with four virtual channels on uniform random traffic with each flit taking 4 cycles to traverse the switch. Other traffic patterns are discussed in Section 7. Packets were injected using a Bernoulli process. The simulator was warmed up under load without taking measurements until steady-state was reached. Then a sample of injected packets were labeled during a measurement interval. The sample size was chosen such that the measurements are accurate to within 3% with 99% confidence. Finally, the simulation was



**Figure 8.** Block diagram of the different VC allocation schemes (a) CVA (b) OVA. In each cycle, CVA can handle multiple VC requests for the same output where as in OVA, only a single VC request for each output can be made. CVA parallelize the switch and VC allocation while in OVA, the two allocation steps are serialized. For simplicity, the logic is shown for only a single output.



**Figure 9.** Latency vs. offered load for the baseline architecture

run until all the labeled packets reached their destinations. We begin the evaluation using single-flit packets; later, we also consider longer packets (10 flits).

A plot of latency versus offered load (as a fraction of the capacity of the switch) is shown in Figure 9. The performance of a low-radix router (radix 16), which follows the pipeline shown in Figure 5(b), with a centralized switch and virtual channel allocation is shown for comparison. Note that this represents an unrealistic design point since the centralized single-cycle allocation does not scale. Even with multiple virtual channels, head-of-line(HoL) blocking limits the low-radix router to approximately 60% throughput [18].

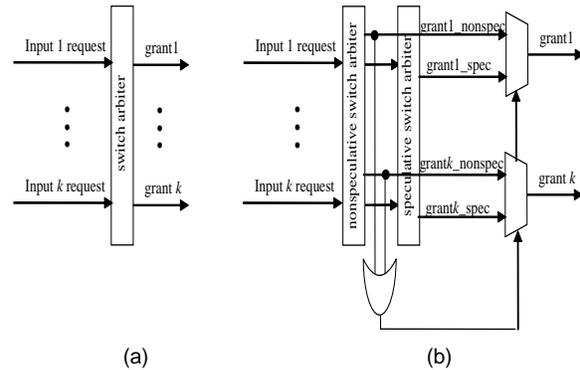
Increased serialization latency gives the high-radix router a higher zero-load latency than the low-radix router when considering only a single stage, as in this case. The saturation throughput of the high-radix router is approximately

50% or 12% lower than the low-radix router. The results reflect the performance of the router with realistic pipeline delays, distributed switch allocation, and a speculative virtual channel allocation. Most of this loss is attributed to the speculative VC allocation. The effect is increased when OVA is used giving a saturation throughput of about 45%.

#### 4.4 Prioritized Virtual Channel Allocation

With speculative VC allocation, if the initial VC allocation fails, bandwidth can be unnecessarily wasted if the re-bidding is not done carefully. For example, consider an input queue with 4 VCs and input arbitration performed in a round-robin fashion. Assume that all of the VCs in the input queues are occupied and the flit at the head of one of the VC queues has failed VC allocation. If all 4 VCs continuously bid for the output one after the other, the speculative bids by the failed VC will waste approximately 25% of the bandwidth until the output VC it is waiting on becomes available.

Bandwidth loss due to speculative VC allocation can be reduced by giving priority in switch allocation to nonspeculative requests [10,27]. This can be accomplished, for example by replacing the single switch allocator of Figure 10(a) with separate switch allocators for speculative and nonspeculative requests as shown in Figure 10(b). With this arrangement, a speculative request is granted bandwidth only if there are no nonspeculative requests. Prioritizing nonspeculative requests in this manner reduces bandwidth loss but at the expense of doubling switch allocation logic.



**Figure 10.** Block diagram of a switch arbiter using (a) one arbiter and (b) two arbiters to prioritize the nonspeculative requests.

In this section we evaluate the performance gained by using two allocators to prioritize nonspeculative requests. The switch simulated in Section 4.3 used only a single switch allocator and did not prioritize nonspeculative requests. To ensure fairness with two switch arbiters, the priority pointer in the speculative switch arbiter is only updated after the speculative request is granted (i.e. when there are no non-

speculative requests). Our evaluation uses only 10-flit packets — with single flit packets, all flits are speculative, and hence there is no advantage to prioritizing nonspeculative flits. We prioritize nonspeculative requests only at the output switch arbiter. Prioritizing at the input arbiter reduces performance by preventing speculative flits representing VC requests from reaching the output virtual channel allocators.

Figure 11 shows that prioritizing nonspeculative requests is advantageous when there is only a single virtual channel, but has little return with four virtual channels. These simulations use CVA for VC allocation. With only a single VC, prioritized allocation increases saturation throughput by 10% and gives lower latency as shown in Figure 11(a). With four VCs, however, the advantage of prioritized allocation diminishes as shown in Figure 11(b). Here the multiple VCs are able to prevent the loss of bandwidth since with multiple VCs, a speculative request will likely find an available output VC. Results for the OVA VC allocation follow the same trend but are not shown for space constraints. Using multiple VCs gives adequate throughput without the complexity of a prioritized switch allocator.

In the following two sections, which introduce two new architectures, we will assume the CVA scheme for VC allocation using a switch allocator without prioritization.

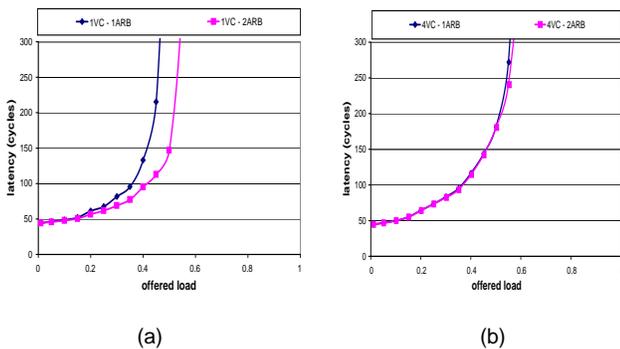


Figure 11. Comparison of using one arbiter and two arbiters for (a) 1VC (b) 4VC

## 5 Buffered Crossbar

Adding buffering at the crosspoints of the switch (Figure 12(b)) decouples input and output virtual channel and switch allocation. This decoupling simplifies the allocation, reduces the need for speculation, and overcomes the performance problems of the baseline architecture with distributed, speculative allocators.

### 5.1 Switch and Virtual Channel Allocation

Input and output switch allocation are completely decoupled. A flit whose request wins the input arbitration is im-

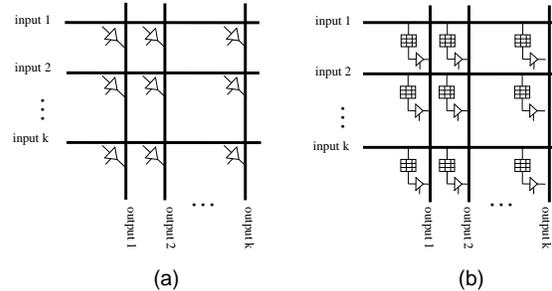


Figure 12. Block diagram of a (a) baseline crossbar switch and (b) fully buffered crossbar switch.

mediately forwarded to the crosspoint buffer corresponding to its output. At the crosspoint, local and global output arbitration are performed as in the unbuffered switch. However, because the flit is buffered at the crosspoint, it does not have to re-arbitrate at the input if it loses arbitration at the output.

The intermediate buffers are associated with the input VCs. In effect, the crosspoint buffers are per-output extensions of the input buffers. Thus, no VC allocation has to be performed to reach the crosspoint — the flit already holds the input VC. Output VC allocation is performed in two stages: a  $v$ -to-1 arbiter that selects a VC at each crosspoint followed by a  $k$ -to-1 arbiter that selects a crosspoint to communicate with the output.

### 5.2 Crosspoint buffer credits

To ensure that the crosspoint buffers never overflow, credit-based flow control is used. Each input keeps a separate free buffer counter for each of the  $k$  crosspoint buffers in its row. For each flit sent to one of these buffers, the corresponding free count is decremented. When a count is zero, no flit can be sent to the corresponding buffer. Likewise, when a flit departs a crosspoint buffer, a credit is returned to increment the input's free buffer count. The required size of the crosspoint buffers is determined by the credit latency — the latency between when the buffer count is decremented at the input and when the credit is returned in an unloaded switch.

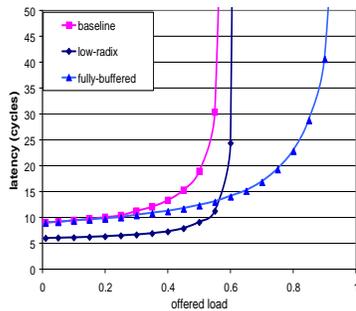
It is possible for multiple crosspoints on the same input row to issue flits on the same cycle (to different outputs) and thus produce multiple credits in a single cycle. Communicating these credits back to the input efficiently presents a challenge. Dedicated credit wires from each crosspoint to the input would be prohibitively expensive. To avoid this cost, all crosspoints on a single input row share a single credit return bus. To return a credit, a crosspoint must arbitrate for access to this bus. The credit return bus arbiter is distributed, using the same local-global arbitration approach as the output switch arbiter.

We have simulated the use of a shared credit return bus

and compared it with an *ideal* (but not realizable) switch in which credits are returned immediately. Simulations show that there is minimal difference between the ideal scheme and the shared bus. The impact of credit return delay is minimized since each flit takes four cycles to traverse the input row. Thus even if a crosspoint loses the credit return bus arbitration, it has 3 additional cycles to re-arbitrate for the bus without affecting the throughput.

### 5.3 Performance and area

We simulated the buffered crossbar using the same simulation setup as described in Section 4.3. In the switch evaluated, each crosspoint buffer contains four flit entries per virtual channel. As shown in Figure 13, the addition of the crosspoint buffers enables a much higher saturation throughput than the unbuffered crossbar while maintaining low latency at low offered loads. This is due both to avoiding head-of-line blocking and decoupling input and output arbitration.

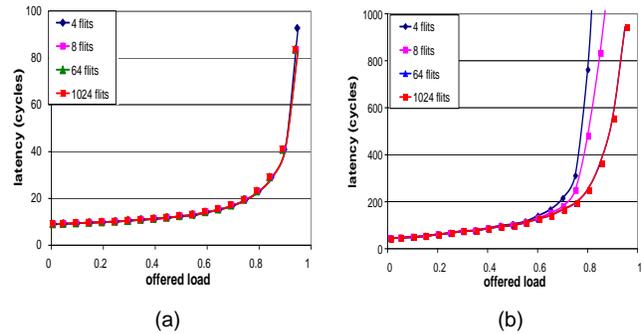


**Figure 13.** Latency vs. offered load for the Fully Buffered Architecture. In both the fully buffered crossbar and the baseline architecture, the CVA scheme is used.

With sufficient crosspoint buffers, this design achieves a saturation throughput of 100% of capacity because the head-of-line blocking is completely removed. As we increase the amount of buffering at the crosspoints, the fully buffered architecture begins to resemble a virtual-output queued (VOQ) switch where each input maintains a separate buffer for each output. The advantage of the fully buffered crossbar compared to a VOQ switch is that there is no need for a complex allocator - the simple distributed allocation scheme discussed in Section 4 is able to achieve 100% throughput.

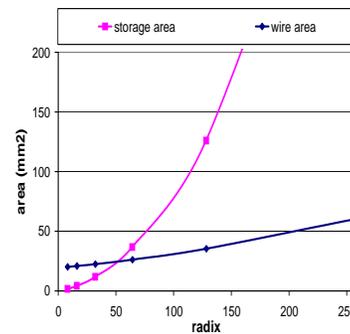
To evaluate the impact of the crosspoint buffer size on performance, we vary the buffer size and evaluate the performance for short and long packets. As shown in Figure 14(a), for short packets four-flit buffers are sufficient to achieve good performance. With long packets, however, larger crosspoint buffers are required to permit enough packets to be

stored in the crosspoint to avoid head-of-line blocking in the input buffers.



**Figure 14.** Latency vs. offered load for the fully buffered architecture for (a) short packet and (b) long packet as the crosspoint buffer size is varied

The performance benefits of a fully-buffered switch come at the cost of a much larger router area. The crosspoint buffering is proportional to  $vk^2$  and dominates chip area as the radix increases. Figure 15 shows how storage and wire area grow with  $k$  in a  $0.10\mu\text{m}$  technology for  $v=4$ . The storage area includes crosspoint and input buffers. The wire area includes area for the crossbar itself as well as all control signals for arbitration and credit return. As radix is increased, the bandwidth of the crossbar (and hence its area) is held constant. The increase in wire area with radix is due to increased control complexity. For a radix greater than 50, storage area exceeds wire area.



**Figure 15.** Area comparison between storage area and wire area in the fully buffered architecture.

### 5.4 Fully Buffered Crossbar without per-VC buffering

One approach to reducing the area of the fully buffered crossbar is to eliminate per-VC buffering at the crosspoints.

With a single shared buffer among the VCs per crosspoint, the total amount of storage area can be reduced by a factor of  $v$ . This approach would still decouple the input and the output switch arbitration, thus providing good performance over a non-buffered crossbar. However, VC allocation is complicated by the shared buffers and it presents new problems.

As discussed in Section 4.2, VC allocation is performed speculatively in order to reduce latency — the flit is sent to the crosspoint without knowing if the output VC can be allocated. With per input VC crosspoint buffers, this was not an issue. However, with a crosspoint buffer shared between input VCs, a flit cannot be allowed to stay in the crosspoint buffer while awaiting output VC allocation. If the speculative flit is allowed to wait in the crosspoint buffer following an unsuccessful attempt at VC allocation, this flit can block all input VCs. This not only degrades performance but also creates dependencies between VCs that may lead to deadlock. Because flits cannot wait for VC allocation in the crosspoint buffers, speculative flits that have been sent to the crosspoint switch must be kept in the input buffer until an ACK is received from output VC allocation. If the flit fails VC allocation or if there are no downstream buffers, the flit is removed from the buffer at the crosspoint and a NACK is sent back to the input, and the input has to resend this flit at a later time. Note that with per-VC buffers, this blocking does not occur and no ACKs are necessary.

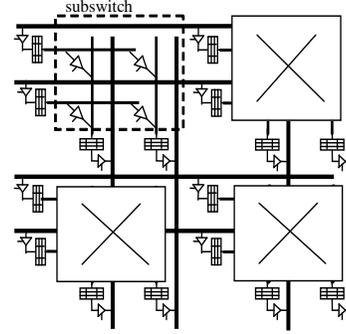
### 5.5 Other Issues

The fully-buffered crossbar presents additional issues beyond the quadratic growth in storage area. This design requires  $k^2$  arbiters, one at each crosspoint, with  $v$  inputs each to arbitrate between the VCs at each crosspoint. In addition, each input needs a  $kv$  entry register file to maintain the credit information for the crosspoint buffers and logic to increment/decrement the credit information appropriately.

Besides the microarchitectural issues, the fully buffered crossbar restricts the routing algorithm that can be implemented. A routing relation may return multiple outputs as a possible next hop. With a fully buffered architecture and the distributed allocators, multiple outputs can not be requested simultaneously and only one output port can be selected. The hierarchical approach that we present in the next section provides a solution that is a compromise between a centralized router and the fully buffered crossbar.

## 6 Hierarchical Crossbar Architecture

A block diagram of the hierarchical crossbar is shown in Figure 16. The hierarchical crossbar is built by dividing the crossbar switch into subswitches where only the inputs and outputs of the subswitch are buffered. A crossbar switch with  $k$  ports that has a subswitch of size  $p$  is made up of  $(k/p)^2$   $p \times p$  crossbars, each with its own input and output buffers.



**Figure 16.** Hierarchical Crossbar ( $k=4$ ) built from smaller subswitches ( $p=2$ ).

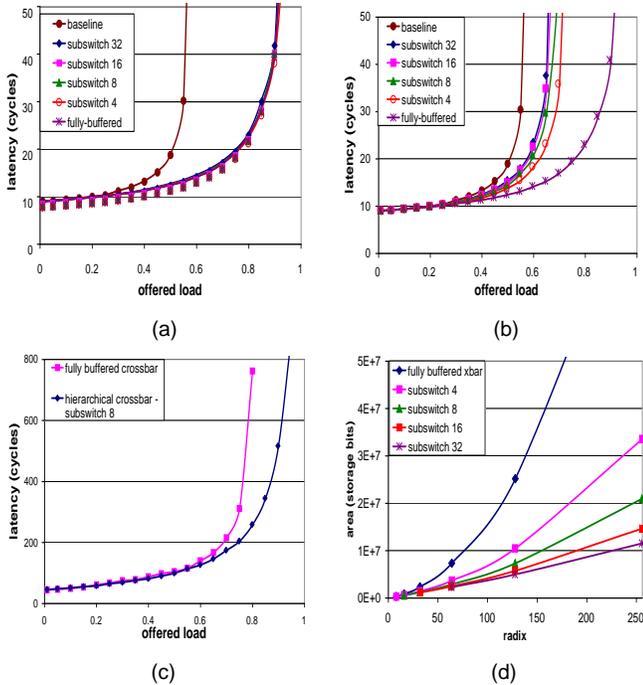
By implementing a subswitch design the total amount of buffer area grows as  $O(vk^2/p)$ , so by adjusting  $p$  the buffer area can be significantly reduced from the fully-buffered design. This architecture also provides a natural hierarchy in the control logic — local control logic only needs to consider information within a subswitch and global control logic coordinates the subswitches.

Similar to the fully-buffered architecture, the intermediate buffers on the subswitch boundaries are allocated on a per-VC basis. The subswitch input buffers are allocated according to a packet's *input* VC while the subswitch output buffers are allocated according to a packet's *output* VC. This decoupled allocation reduces HoL blocking when VC allocation fails and also eliminates the need to NACK flits in the intermediate buffers. By having this separation at the subswitches with buffers, it divides the VC allocation into a local VC allocation within the subswitch and a global VC allocation among the subswitches.

With the hierarchical design, an important design parameter is the size of the subswitch,  $p$  which can range from 1 to  $k$ . With small  $p$ , the switch resembles a fully-buffered crossbar resulting in high performance but also high cost. As  $p$  approaches the radix  $k$ , the switch resembles the baseline crossbar architecture giving low cost but also lower performance.

The throughput and area of hierarchical crossbars with various subswitch sizes are compared to the fully buffered crossbar and the baseline architecture in Figure 17. On uniform random traffic (Figure 17(a)), the hierarchical crossbar performs as well as the fully buffered crossbar, even with a large subswitch size. With uniform random traffic, each subswitch see only a fraction of the load —  $\frac{\lambda}{k/p}$  where  $\lambda$  is the total offered load. Even with just two subswitches, the maximum load seen by any subswitch for uniform random traffic pattern will always be less than 50% and the subswitches will not be saturated.

A worst-case traffic pattern for the hierarchical crossbar



**Figure 17.** Comparison of the hierarchical crossbar as the subswitch size is varied (a) uniform random traffic (b) worst-case traffic (c) long packets and (d) area.  $k=64$  and  $v=4$  is used for the comparison.

concentrates traffic on a small number of subswitches. For this traffic pattern, each group of  $(k/p)$  inputs that are connected to the same row of subswitches send packets to a randomly selected output within a group of  $(k/p)$  outputs that are connected to the same column of subswitches. This concentrates all traffic into only  $(k/p)$  of the  $(k/p)^2$  subswitches. Figure 17(b) shows performance on this traffic pattern. The benefit of having smaller subswitch size is apparent. On this worst-case pattern, the hierarchical crossbar does not achieve the throughput of the fully-buffered crossbar (about 30% less throughput for  $p = 8$ ). However hierarchical crossbars outperforms the baseline architecture by 20% (for  $p = 8$ ). Fortunately, this worst-case traffic pattern is very unlikely in practice.

Like the fully-buffered crossbar, the throughput of the hierarchical crossbar on long packets depends on the amount of intermediate buffering available. The evaluation so far assumed that each buffer in the hierarchical crossbar holds four flits. In order to provide a fair comparison, we keep the *total* buffer size constant and compare the performance of the hierarchical crossbar with the fully buffered crossbar on long packets. The fully buffered crossbar has 4 entries per crosspoint buffer while the hierarchical crossbar ( $p = 8$ ) has 16 en-

tries per buffer.<sup>6</sup> Figure 17(c) compares the performance of a fully-buffered crossbar with a hierarchical crossbar ( $p = 8$ ) with equal total buffer space. Under this constraint, the hierarchical crossbar provides better throughput on uniform random traffic than the fully-buffered crossbar.

The cost of the two architectures, in terms of area, is compared in Figure 17(d). The area is measured in terms of the storage bits required in the architecture. As radix increases, there is quadratic growth in the area consumed by the fully buffered crossbar. For  $k = 64$  and  $p = 8$ , a hierarchical crossbar takes 40% less area than a fully-buffered crossbar.

## 7 Simulation Results

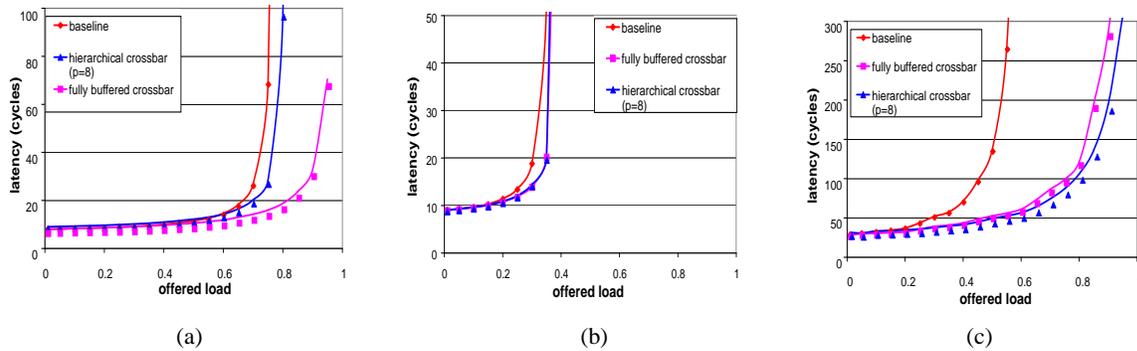
In addition to uniform random traffic, we present additional simulations to compare the architectures presented using traffic patterns summarized in Table 1. The results of the simulations are shown in Figure 18. On diagonal traffic, the hierarchical crossbar exceeds the throughput of the baseline by 10%. Hotspot traffic limits the throughput to under 40% capacity for all three architectures. At this point the oversubscribed outputs are saturated. The hierarchical crossbar and the fully-buffered crossbar achieve nearly 100% throughput on bursty traffic while the baseline architecture saturates at 50%. The hierarchical crossbar outperforms the full-buffered crossbar on this pattern. It is better able to handle bursts of traffic because it has two stages of buffering, at both the inputs and the outputs of each subswitch, even though it has less total buffering than the fully-buffered crossbar.

Name	Description
diagonal traffic	traffic pattern where input $i$ send packets only to output $i$ and $(i + 1) \bmod k$
hotspot	uniform traffic pattern with $h = 8$ outputs being oversubscribed. For each input, 50% of the traffic is sent to the $h$ outputs and the other 50% is randomly distributed.
bursty	uniform traffic pattern is simulated with a bursty injection based on a Markov ON/OFF process and average burst length of 8 packets is used.

**Table 1.** Nonuniform traffic pattern evaluated.

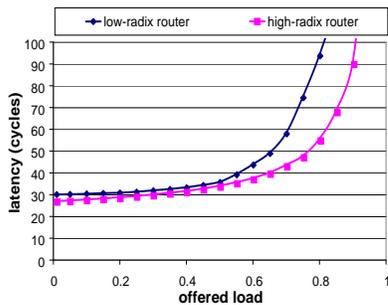
While a single high-radix router has higher zero-load latency than a low-radix router (Figure 9), this factor is more than offset by the reduced hop-count of a high-radix network giving lower zero-load latency for the network as a

<sup>6</sup>To make the total buffer storage equal, each input and output buffer in the hierarchical crossbar has  $p/2$  times the storage of a crosspoint buffer in the fully-buffered crossbar.



**Figure 18.** Performance Comparison on Nonuniform Traffic pattern (a) diagonal traffic (b) hotspot traffic (c) bursty traffic. Parameters used are  $k=64$ ,  $v=4$ , and  $p=8$  with 1 flit packets

whole. Latency as a function of the offered load for a network of 4096 nodes with both radix-64 and radix-16 routers is shown in Figure 19. Both routers use the hierarchical architecture proposed in Section 6. The routers are configured as a Clos [6] network with three stages for the radix-64 routers and five stages for the radix-16 routers. The simulation was run using an oblivious routing algorithm (middle stages are selected randomly) and uniform random traffic. Because of the complexity of simulating a large network, we use the simulation methodology outlined in [19] to reduce the simulation time with minimal loss in the accuracy of the simulation.



**Figure 19.** Network simulation comparison

## 8 Related Work

Most existing single chip router architectures are designed for small radix implementations [24, 30]. Commodity routers such as Quadrics [3] implement a radix-8 router and the highest radix available from Myrinet is radix-32 [25]. The IBM SP2 switch [32] is a radix-8.

The scaling issue of switches have been addressed in IP routers in order to support the increasing line rate. The IBM Prizma architecture third generation switch [12] has increased the number of ports from 32 to 64. To overcome

the limitation of the scheduling or the allocation in these IP routers, buffered crossbars [29] have been studied which decouple the input and the output allocation and has been shown to achieve high performance. However, the routers for these switch fabrics are fundamentally different. IP routers have packets that are significant longer (usually at least 40B to 100B) compared to the packet size of a shared memory multiprocessors (8B to 16B). Thus, IP routers are less sensitive to latency in the design than the high-radix routers used in a multi-computer system. In addition, these IP routers are often built using multiple chips and thus, do not have the area constraint that is present in our design.

High-radix crossbars have been previously designed using multiple lower-radix crossbars. A design implemented with multiple chips and each chip acting as a sub-crossbar is outlined in [10]. Other work has attempted to exploit traffic characteristics to partition the crossbar into smaller, faster subcrossbars [5] but does not scale to high radix. A two-dimensional crossbar with VOQs has been decomposed to a switch organization similar to our hierarchical router [17]. However, these references neither discuss how to scale the allocation schemes to high-radix nor do they provide the per-VC intermediate buffering at the subswitches.

To prevent HoL blocking, virtual output queueing (VOQ) is often used in IP routers where each input has a separate buffer for each output [23]. VOQ adds  $O(k^2)$  buffering and becomes costly, especially as  $k$  increases. To overcome the area limitation and fit on a single chip, we present an alternate placement of the buffers at subswitches in a hierarchical implementation of the crossbar. Hierarchical arbitration has been previously studied and the arbitration logic used in this work resembles the earlier work from [4]. The novelty of the hierarchical crossbar is in the placement of buffers at the input and outputs of the subswitch and in the decoupling of the virtual channel allocation.

## 9 Conclusion and Future Work

To exploit advances in technology, high-radix routers are needed to convert available chip bandwidth into lower latency and cost. These routers use a larger number of ports with narrower channels instead of a smaller number of ports with wider channels. Existing microarchitectures for building routers do not scale to high radix. Naive scaling of a baseline architecture provides a simple design but provides less than 50% throughput. A fully-buffered crossbar with per-VC buffering at the crosspoints provides nearly 100% throughput but at a prohibitive cost. We propose an alternative architecture, the hierarchical crossbar, that maintains high performance but at a lower, realizable cost. The hierarchical crossbar provides a 20-60% increase in the throughput over the baseline architecture and results in a 40% area savings compared to the fully buffered crossbar. The hierarchical nature of the architecture provides the benefit of logically separating the control logic in a hierarchical manner as well.

The migration to high-radix routers opens many opportunities for future work. High-radix routers reduce network hop count, presenting challenges in the design of *optimal* network topologies. New routing algorithms are required to deal both with the large number of ports on each router and with new topologies that we expect to emerge to best exploit these routers. In this work, we only consider crossbar switch architectures. Alternative internal switch organizations (e.g., on chip networks with direct or indirect topologies) can potentially reduce implementation costs further and enable scaling to very high radices.

## Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments. This work has been supported in part by a grant from the Stanford Networking Research Center, in part by an NSF Graduate Fellowship, and in part by New Technology Endeavors, Inc. through DARPA sub-contract CR03-C-0002 under U.S. government Prime Contract Number NBCH3039003.

## References

- [1] A. Agarwal. Limits on Interconnection Network Performance. *IEEE Trans. Parallel Distrib. Syst.*, 2(4):398–412, 1991.
- [2] A. Agarwal et al. The MIT Alewife Machine: Architecture and Performance. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture*, pages 2–13, 1995.
- [3] J. Beecroft, D. Addison, F. Petrini, and M. McLaren. Quadrics QsNet II: A Network for Supercomputing Applications. In *Hot Chips 15*, Stanford, CA, August 2003.
- [4] H. J. Chao, C. H. Lam, and X. Guo. A Fast Arbitration Scheme for Terabit Packet Switches. In *Proc. of IEEE Global Telecommunications Conf.*, pages 1236–1243, 1999.
- [5] Y. Choi and T. M. Pinkston. Evaluation of Crossbar Architectures for Deadlock Recovery Routers. *J. Parallel Distrib. Comput.*, 61(1):49–78, 2001.
- [6] C. Clos. A Study of Non-Blocking Switching Networks. *The Bell System technical Journal*, 32(2):406–424, March 1953.
- [7] Cray X1. <http://www.cray.com/products/systems/x1/>.
- [8] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.
- [9] W. J. Dally and C. L. Seitz. The Torus Routing Chip. *Distributed Computing*, 1(4):187–196, 1986.
- [10] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, San Francisco, CA, 2004.
- [11] T. Dunigan. Early Experiences and Performance of the Intel Paragon. *Technical report, Oak Ridge National Laboratory, ORNL/TM-12194.*, 1993.
- [12] A. P. J. Engbersen. Prizma Switch Technology. *IBM J. Res. Dev.*, 47(2-3):195–209, 2003.
- [13] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren. Architecture and Design of AlphaServer GS320. In *Proc. of the 9th Int'l conf. on Architectural support for programming languages and operating systems*, pages 13–24, 2000.
- [14] F. Heaton, B. Dally, W. Dettloff, J. Eyles, T. Greer, J. Poulton, T. Stone, and S. Tell. A Single-Chip Terabit Switch. In *Hot Chips 13*, Stanford, CA, 2001.
- [15] M. Horowitz, C. K. K. Yang, and S. Sidiropoulos. High-Speed Electrical Signaling: Overview and Limitations. *IEEE Micro*, 18(1):12–24, 1998.
- [16] IBM Redbooks. *An Introduction to the New IBM eServer pSeries High Performance Switch*.
- [17] J. Jun, S. Byun, B. Ahn, S. Y. Nam, and D. Sung. Two-Dimensional Crossbar Matrix Switch Architecture. In *Asia Pacific Conf. on Communications*, pages 411–415, September 2002.
- [18] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input versus Output Queueing on a Space-division Packet Switch. *IEEE Transactions on Communications*, COM-35(12):1347–1356, 1987.
- [19] J. Kim, W. J. Dally, and A. K. Gupta. Simulation of a High-radix Interconnection Network. *CVA Technical Report*, March 2005.
- [20] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture*, pages 241–251, 1997.
- [21] M. J. E. Lee, W. J. Dally, R. Farjad-Rad, H.T. Ng, R. Senthinathan, J. H. Edmondson, and J. Poulton. CMOS High-Speed I/Os - Present and Future. In *International Conf. on Computer Design*, pages 454–461, San Jose, CA, 2003.
- [22] C. E. Leiserson et al. The Network Architecture of the Connection Machine CM-5. *J. Parallel Distrib. Comput.*, 33(2):145–158, 1996.
- [23] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% Throughput in an Input Queue Switch. In *Proc. IEEE INFOCOM'96.*, pages 296–302., San Francisco, CA, 1996.
- [24] S. Mukherjee, P. Bannan, S. Lang, A. Spink, and D. Webb. The Alpha 21364 network architecture. In *Hot Chips 9*, pages 113–117, Stanford, CA, August 2001.
- [25] Myrinet. <http://www.myricom.com/myrinet/overview/>.
- [26] M. D. Noakes, D. A. Wallach, and W. J. Dally. The J-Machine Multi-computer: An Architectural Evaluation. In *Proc. of the 20th Annual Int'l Symp. on Computer Architecture*, pages 224–235, 1993.
- [27] L. S. Peh and W. J. Dally. A Delay Model for Router Micro-architectures. *IEEE Micro*, 21(1):26–34, 2001.
- [28] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, 2002.
- [29] R. Rojas-Cessa, E. Oki, and H. Chao. CIXOB-k: Combined Input-Crosspoint-Output Buffered Packet Switch. In *Proc. of IEEE Global Telecom. Conf.*, pages 2654–2660, San Antonio, TX, 2001.
- [30] S. Scott and G. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Hot Chips 4*, Stanford, CA, Aug. 1996.
- [31] SGI Altix 3000. <http://www.sgi.com/products/servers/altix/>.
- [32] C. B. Stunkel et al. The SP2 High-performance Switch. *IBM Syst. J.*, 34(2):185–204, 1995.
- [33] H. Wang, L. S. Peh, and S. Malik. Power-driven Design of Router Microarchitectures in On-chip Networks. In *Proc. of the 36th Annual IEEE/ACM Int'l Symposium on Microarchitecture*, pages 105–116, 2003.