2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip

Design of a High-Throughput Distributed Shared-Buffer NoC Router

Rohit Sunkam Ramanujam*, Vassos Soteriou†, Bill Lin* and Li-Shiuan Peh‡
*Department of Electrical and Computer Engineering, University of California, San Diego
†Department of Electrical Engineering and Information Technology, Cyprus University of Technology
‡Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology

***Abstract*—Router microarchitecture plays a central role in the performance of an on-chip network (NoC). Buffers are needed in routers to house incoming flits which cannot be immediately forwarded due to contention. This buffering can be done at the inputs or the outputs of a router, corresponding to an input-buffered router (IBR) or an output-buffered router (OBR). OBRs are attractive because they can sustain higher throughputs and have lower queuing delays under high loads than IBRs. However, a direct implementation of an OBR requires a router speedup equal to the number of ports, making such a design prohibitive under aggressive clocking needs and limited power budgets of most NoC applications. In this paper, we propose a new router design that aims to emulate an OBR practically, based on a distributed shared-buffer (DSB) router architecture. We introduce innovations to address the unique constraints of NoCs, including efficient pipelining and novel flow-control. We also present practical DSB configurations that can reduce the power overhead with negligible degradation in performance. The proposed DSB router achieves upto 19% higher throughput on synthetic traffic and reduces packet latency by 60% on average for SPLASH-2 benchmarks with high contention, compared to a state-of-art pipelined IBR. On average, the saturation throughput of DSB routers is within 10% of the theoretically ideal saturation throughput under the synthetic workloads evaluated.**

***Keywords*-On-chip interconnection networks, Router microarchitecture.**

I. Introduction

Network-on-Chip (NoC) architectures are becoming the de facto fabric for both general-purpose chip multi-processors (CMPs) and application-specific systems-on-chips (SoCs). In the design of NoCs, high throughput and low latency are both important design parameters and the router microarchitecture plays a vital role in achieving these performance goals. High throughput routers allow an NoC to satisfy the communication needs of multi- and many-core applications, or the higher achievable throughput can be traded off for power savings with fewer resources being used to attain a target bandwidth. Further, achieving high throughput is also critical from a delay perspective for applications with heavy communication workloads because queueing delays grow rapidly as the network approaches saturation.

A router's role lies in efficiently multiplexing packets onto the network links. Router buffering is used to house arriving flits[1] that cannot be immediately forwarded to the output links due to contention. This buffering can be done either at the inputs or the outputs of a router, corresponding to an input-buffered router (IBR) or an output-buffered router (OBR). OBRs are attractive for NoCs because they can sustain higher throughputs and have lower queueing delays under high loads than IBRs. However, a direct implementation of an OBR requires each router to operate at P times speedup, where P is the number of router ports. This can either be realized with the router being clocked at P times the link clock frequency, or the router having P times more internal buffer and crossbar ports. Both of these approaches are prohibitive given the aggressive design goals of most NoC applications, such as high-performance CMPs. This is a key reason behind the broad adoption of IBR microarchitectures as the preferred design choice and the extensive prior effort in the computer architecture community on aggressively pipelined IBR designs.

In this paper, we propose a new router microarchitecture that aims to emulate an OBR without the need for any router speedup. It is based on a distributed shared-buffer (DSB) router architecture that has been successfully used in high-performance Internet packet routers [4], [15]. Rather than buffering data at the output ports, a DSB router uses two crossbar stages with buffering sandwiched in between. These buffers are referred to as middle memories. To emulate the first-come, first-served (FCFS) order of an output-buffered router, incoming packets are timestamped with the same departure times as they would depart in an OBR. Packets are then assigned to one of the middle memory buffers with two constraints. First, packets that are arriving at the same time must be assigned to different middle memories. Second, an incoming packet cannot be assigned to a middle memory that already holds a packet with the same departure time[2]. It has been shown in [4], [15] that for a P port router, $N \geq (2P-1)$ middle memories are necessary and sufficient to ensure that memory assignments are always possible. This scheme emulates a FCFS output-buffered router if unlimited buffering is available.

However, just as the design objectives and constraints for an on-chip IBR are quite different from those for an Internet packet router, the architecture tradeoffs and design constraints for an on-chip DSB router are also quite different. First, limited power and area budgets restrict practical implementations to small amounts of buffering. This makes it imperative to

A short version of this paper appeared in the IEEE Computer Architecture Letters [16].

[1]A flit is a fixed-size portion of a packetized message.

[2]This is necessary to avoid switch contention.

explore power and area efficient DSB configurations suitable for being implemented on a chip. Next, a flow-control protocol which can work with few buffers is necessary since NoC applications such as cache coherence protocols cannot tolerate dropping of packets. A novel flow-control scheme is also needed for ultra-low latency communication in NoCs for supporting a wide range of delay-sensitive applications. Unlike Internet routers that typically use store-and-forwarding of packets, flit-level flow-control is widely used in on-chip routers where bandwidth and storage are allocated at the level of flits. Finally, another key difference is the need for on-chip routers to operate at aggressive clock frequencies. This makes it important to have an efficient router pipeline where delay and complexity are balanced across all pipeline stages. Our proposed router microarchitecture tackles all these challenges with novel designs.

The remainder of this paper is organized as follows. Section II provides background information on throughput analysis and on existing router architectures. Section III describes our proposed distributed shared-buffer router microarchitecture for NoCs. Next, Section IV provides extensive throughput and latency evaluations of our proposed DSB architecture using a detailed cycle-accurate simulator on a range of synthetic network traces and traffic traces gathered from real system simulations, while Section V evaluates the power and area overhead of DSB routers. Section VI reviews related work. Finally, Section VII concludes the paper.

II. Background

We first provide a brief background on throughput analysis. We then present a short description of OBR and IBR microarchitectures, focusing on their deficiencies in practically attaining ideal throughput, before discussing distributed-shared-buffer Internet routers and how they mimic output buffering.

A. Throughput Analysis

In this section, we provide a brief overview of the analysis techniques used to evaluate ideal network throughput. In particular, we elaborate on the concepts of network capacity, channel load, and saturation throughput. These concepts are intended to capture what could be *ideally* achieved for a routing algorithm R on a given traffic pattern Λ. To decouple the effects of the router microarchitecture, including buffer sizing and the flow-control mechanism being utilized, ideal throughput analysis is based on channel load analysis. We first review the concept of network capacity.

Network capacity: Network capacity is defined by the maximum channel load γ^* that a channel at the bisection of the network needs to sustain under uniformly distributed traffic. As shown in [2], for any $k \times k$ mesh,

$$\gamma^* = \begin{cases} \frac{k}{4} & \text{for even } k \\ \frac{k^2-1}{4k} & \text{for odd } k \end{cases}$$

The network capacity, $\mathcal{N}$, in flits per node per cycle is then defined as the inverse of γ^*:

$$\mathcal{N} = \frac{1}{\gamma^*} = \begin{cases} \frac{4}{k} & \text{for even } k \\ \frac{4k}{k^2-1} & \text{for odd } k \end{cases}$$

TABLE I
Traffic patterns and their corresponding ideal saturation throughput under dimension-ordered XY routing.

Traffic	Description	Saturation throughput
Uniform	Destination chosen at random, uniformly	1.0
Tornado	(x,y) to $((x+\lceil\frac{k}{2}\rceil-1)\%k,\ (y+\lceil\frac{k}{2}\rceil-1)\%k)$	0.66
Complement	(x,y) to $(k-x-1,\ k-y-1)$	0.5

For example, for a $k \times k$ mesh, with $k = 8$, $\mathcal{N} = 4/8 = 0.5$ flits/node/cycle. Next, we review the concept of saturation throughput.

Saturation throughput: For a routing algorithm R and a given traffic pattern Λ, the expected *channel load* on a channel c is denoted as $\gamma_c(R,\Lambda)$. The *normalized worst-case channel load*, $\gamma_{wc}(R,\Lambda)$, is then defined as the expected number of flits crossing the most heavily loaded channel, normalized to γ^*.

$$\gamma_{wc}(R,\Lambda) = \frac{\max_{c\in C}\gamma_c(R,\Lambda)}{\gamma^*}$$

where C is the set of all channels in the network.

Given this definition of normalized worst-case channel load, the *saturation throughput* corresponds to the average number of flits that can be injected by all the nodes in the network per cycle so as to saturate the most heavily loaded channel to its unit capacity. This is given as:

$$\Theta(R,\Lambda) = \frac{1}{\gamma_{wc}(R,\Lambda)}$$

Saturation throughput is defined specifically for a given routing algorithm R and traffic pattern Λ. Table I shows a few commonly used traffic patterns and their corresponding saturation throughput under dimension-ordered XY routing (DOR-XY). Note that 100% capacity cannot always be achieved with DOR-XY routing even under an *ideal* router design, defined as the one that can handle injection loads up to the saturation throughput. For example, for an adversarial traffic pattern like bit-complement traffic, it is well-known that DOR-XY routing saturates at 50% of network capacity. To decouple the effects of the chosen routing algorithm on network performance, we assume DOR-XY routing throughout the remainder of this paper. The goal of our router design is to reach the ideal router performance and thus approach the achievable saturation throughput.

B. Output-buffered routers

Fact 1: An OBR with unlimited buffering can achieve the theoretical saturation throughput.

Fact 2: OBRs with unlimited buffering have predictable and bounded packet delays when the network is below saturation.

Emulating the first-come, first-served (FCFS) behavior of OBR architectures is important for exploiting their attractive high-throughput and low-delay properties. Throughput guarantees offered by all oblivious routing algorithms [2], which are often used in NoCs because of their simplicity, assume ideal output-buffered routing with infinite buffers in their throughput analysis. When the network topology and the traffic matrix are both known, the saturation throughput for oblivious routing algorithms can be computed based on worst-case channel load analysis (as described in Section II-A). Even when no information about the spatial characteristics of the traffic is available, which is often the case, worst-case throughput guarantees can be provided for oblivious routing functions by solving bipartite maximum-weight matching problems for each channel [20]. These throughput guarantees do not hold if the routers used do not emulate an OBR. Generally, using IBRs, the worst-case saturation throughput of an oblivious routing algorithm can be quite far off from the value predicted by worst-case throughput analysis. So one key advantage of OBR emulation is to provide and retain such guarantees with the limited hardware resources available in on-chip routers. OBRs also have lower and more predictable queueing delays than IBRs because of their FCFS servicing scheme. Flits are not delayed in OBRs unless the delay is unavoidable due to multiple flits arriving at the same time at different inputs destined for the same output. On the other hand, the switch arbitration schemes used in IBRs for multiplexing packets onto links are sub-optimal and result in unpredictable packet delays. The predictability of packet delays is an important concern for delay-sensitive NoC applications and OBR emulation is a step forward in this direction.

Figure 1 depicts the OBR architecture. In this architecture, incoming flits are directly written into the output buffers through a concentrator. Since up to P flits may arrive together for a particular output in the same cycle, a direct implementation of an OBR would require a router speedup of P, where P is the number of router ports (i.e., $P = 5$ in Figure 1). Router speedup can be realized in two ways. First, by clocking the router at P times the link frequency, which is highly impractical with today's aggressive clock rates. Even if realizable, this will lead to exorbitant power consumption. Second, it can be realized with higher internal port counts at the buffers and the crossbar: each output buffer needs P write ports, with input ports connected to the output buffers through a $P \times P^2$ crossbar. This scenario leads to huge CMOS area penalties. High power and area requirements for OBR implementation are the key reasons behind the broad adoption of IBR microarchitectures as the principal design choice and the extensive prior effort in the computer architecture community on aggressively pipelined IBR designs (see Section VI), despite the very attractive property that an OBR can theoretically reach the ideal saturation throughput. Subsequently, in Section II-D we review a distributed shared-buffer (DSB) router architecture that emulates an OBR, inheriting its elegant theoretical properties, without the need for P times router speedup. We first review the IBR microarchitecture that is widely used in on-chip interconnection networks.

C. Input-buffered router microarchitecture

Figure 2-A sketches a typical input-buffered router (IBR) microarchitecture [2] that is governed by virtual channel flow-control [1]. We adopt this as the baseline input-buffered router for our evaluations. The router has P ports, where P depends on the dimension of the topology. In a 2-dimensional mesh,

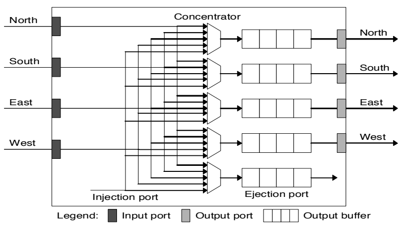


Fig. 1. Output-buffered router (OBR) architecture model.

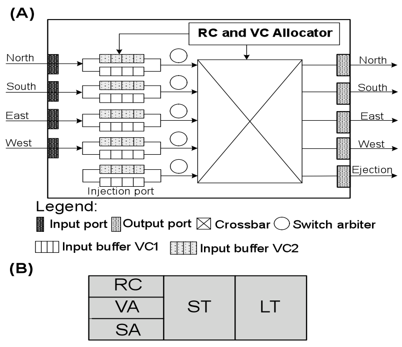


Fig. 2. (A) A typical input-buffered router (IBR) microarchitecture with virtual channel flow-control and 2 virtual channels (VCs) and (B) its 3-stage pipeline (1) route computation (RC) + virtual channel allocation (VA) + switch arbitration (SA), (2) switch traversal (ST) and (3) link traversal (LT).

$P = 5$ as it includes the 4 NSEW ports as well as the injection/ejection port from/to the processor core. At each input port, buffers are organized as separate FIFO queues, one for each virtual channel (VC). Flits entering the router are placed in one of these queues depending on their virtual channel ID. Queues of an input port typically share a single crossbar input port, as shown in Figure 2-A, with crossbar output ports directly connected to output links that interconnect the current (upstream) router with its neighboring (downstream) router.

Figure 2-B shows the corresponding IBR router pipeline. The router uses look-ahead routing and speculative switch allocation resulting in a short three-stage router pipeline. Route computation (RC) determines the output port based on the packet destination and is done one hop in advance. Speculative switch allocation (SA) is done in parallel with VC allocation (VA) and priority is given to non-speculative switch requests to ensure that performance is not hurt by speculation. Once SA and VA are completed, flits traverse the crossbar (ST) before finally traversing the output link (LT) towards the downstream router. Head flits proceed through all the stages while the body and tail flits skip the RC and VA stages and inherit the VC allocated to the head flit. The tail flit releases the reserved VC after departing the upstream router.

To attain high throughput, an IBR relies on several key microarchitectural components to effectively multiplex flits onto the output links. First, additional buffering allows a greater number of flits to be housed at a router during high contention. This increases the number of competing flit candidates to eventually traverse a link towards a downstream router, while also alleviating congestion at upstream routers. Second, a higher number of VCs allows a greater number of individual packet flows to be accommodated, increasing buffer

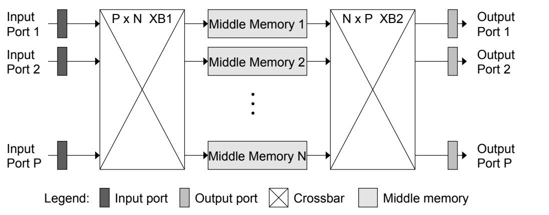


Fig. 3. Distributed shared-buffer router architecture model.

utilization and ultimately link utilization. Finally, the VC and switch allocators need to have good matching capability, i.e. they should ensure that VCs and crossbar switch ports never go idle when there are flits waiting for them. However, with matching algorithms that can be practically implemented in hardware, the throughput of IBRs can be quite far off from the ideal saturation throughput.

D. Distributed shared-buffer routers

DSB routers have been successfully used in high-performance Internet packet routers [4], [15] to emulate OBRs without any internal router speedup. Rather than buffering data directly at the output ports, a DSB router uses two crossbar stages with buffering sandwiched in between. Figure 3 depicts the DSB microarchitecture used in Internet routers. The input ports are connected via a $P \times N$ crossbar to N middle memories. These N middle memories are then connected to the output ports through a second $N \times P$ crossbar. Every cycle, one packet can be read from and written to each middle memory. It should be noted here that when contention occurs in Internet routers, packets can be dropped. This is not allowed in our proposed on-chip DSB architecture (see Section III).

To emulate the first-come, first-served (FCFS) packet servicing in OBRs, a DSB router has to satisfy two conditions: (a) a packet is dropped by the DSB router if and only if it will also be dropped by an OBR, and (b) if a packet is not dropped, then the packet must depart the DSB router at the same cycle as the cycle in which it would have departed an OBR. To achieve this emulation, each packet arriving at a DSB router is *timestamped* with the cycle in which it would have departed from an OBR (i.e., in FCFS order). When a packet arrives, a scheduler is needed to choose a middle memory to which to write this incoming packet and to configure the corresponding first crossbar. Also, at each cycle, packets whose timestamp is equal to the current cycle are read from the middle memories and transferred to the outputs through the second crossbar.

In [4], [15], it was shown that a middle memory assignment can always be found and a DSB router can exactly emulate an OBR if the following condition is satisfied:

Fact 3: A distributed shared-buffer router with $N \geq (2P - 1)$ middle memories and unlimited buffering in each can exactly emulate a FCFS OBR with unlimited buffering.

At least $(2P-1)$ middle memories are needed to ensure that two types of conflicts can always be resolved: The first type of conflict is an *arrival conflict*. A packet has an arrival conflict with all packets that arrive simultaneously at other input ports since packets arriving at the same time cannot be written to the same middle memory. With P input ports, the maximum number of arrival conflicts a packet can have is $(P-1)$. The second type of conflict is a *departure conflict*. A packet has a departure conflict with all other packets that have the same timestamp and need to depart simultaneously through different output ports. With P output ports, a packet can have departure conflicts with at most $(P-1)$ other packets. Therefore, by the pigeonhole principle, $N \geq (2P-1)$ middle memories are necessary and sufficient to find a conflict-free middle memory assignment for all incoming packets.

III. Proposed Distributed Shared Buffer (DSB) Router for NoCs

A. Key architectural contributions

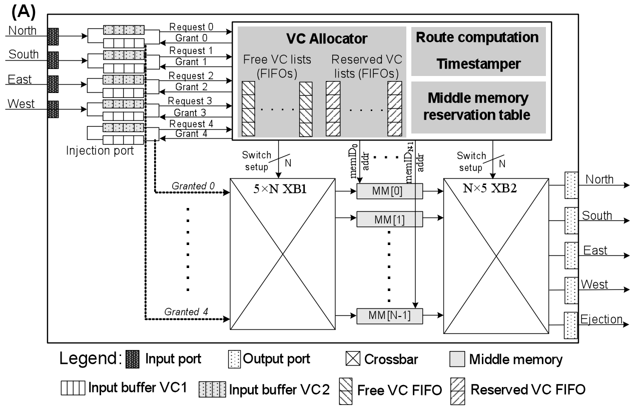
The proposed DSB NoC router architecture addresses the bottlenecks that exist in the data path of IBRs which lead to lower than theoretical ideal throughput. At the same time, it tackles the inherent speedup limitations and area penalties of OBRs while harnessing their increased throughput capabilities. The middle memories decouple input virtual channel queueing from output channel bandwidth, as any flit can acquire any middle memory provided that there are no timing conflicts with other flits already stored in the same middle memory. Essentially, middle memories provide path diversity between the input and output ports within a router. Although based on the DSB architecture used in Internet routers, the proposed NoC router architecture faces a number of challenges specific to the on-chip domain.

First and foremost, NoC applications such as cache coherence protocols cannot tolerate dropping of packets unlike Internet protocols. As a result, the DSB architecture used in Internet routers cannot be directly applied to the on-chip domain. To guarantee packet delivery, a flow control mechanism needs to be in place. The proposed DSB router uses credit-based flit-level flow control. To implement credit-based flow control, we introduce input buffers with virtual channels and distribute the available router buffers between the input ports and the middle memories. Flow-control is applied on a flit-by-flit basis, advancing each flit from an input queue towards any time-compatible middle memory and ultimately to the output link. Flits are timestamped and placed into a middle memory only when the next-hop router has buffers available at its corresponding input port. Further, since the middle memory buffering is limited due to power and area constraints, flits are held back in the input buffers when they fail to find a compatible middle memory.

Next, since power is of utmost importance in the NoC domain, the power-performance tradeoff of different DSB configurations need to be explored. Although, theoretically, $2P-1$ middle memories are needed in a P-port router to avoid all possible arrival and departure conflicts, having a large number of middle memories increases power and area overheads by increasing the crossbar size. Therefore, we evaluate DSB configurations with fewer than $2P-1$ middle memories and estimate its impact on performance.

Finally, on-chip routers need to operate at aggressive clock frequencies, pointing to the need for careful design of router pipelines with low complexity logic at each stage. Our design assumes a delay and complexity-balanced 5-stage pipeline.

(A)



(B)

RC	CR	XB1 + MM_WR	MM_RD + XB2	LT
TS	VA			

Fig. 4. Distributed shared-buffer (A) router microarchitecture with N middle memories and (B) its 5-stage pipeline: (1) Route computation (RC) + timestamping (TS), (2) conflict resolution (CR) and virtual channel allocation (VA), (3) first crossbar traversal (XB1) + middle memory write (MM_WR), (4) middle memory read (MM_RD) and second crossbar traversal (XB2), and (5) link traversal (LT).

The proposed DSB architecture can achieve much higher performance than virtual-channel IBRs with comparable buffering while adding reasonable power and area overheads in managing middle memories and assigning timestamps to flits.

B. DSB microarchitecture and pipeline

Figure 4 shows the router microarchitecture of the proposed DSB router and its corresponding 5-stage pipeline. Incoming flits are first buffered in the input buffers which are segmented into several atomic virtual channels (VCs). The route computation stage (RC) employs a look-ahead routing scheme like the baseline IBR architecture, where the output port of a packet is computed based on the destination coordinates, one hop in advance. This is done only for the head flit of a packet. The remaining pipeline stages of a DSB router are substantially different from those of IBRs. Instead of arbitrating for free virtual channels (buffering) and passage through the crossbar switch (link), flits in a DSB router compete for two resources: middle memory buffers (buffering) and a unique time at which to depart from the middle memory to the output port (link).

The timestamping stage (TS) deals with the timestamp resource allocation. A timestamp refers to the future cycle in which a flit will be read from a middle memory, through the second crossbar, XB2, onto the output port. Timestamping is carried out in conjunction with lookahead routing. A flit (head, body or tail) enters the TS stage and issues a request to the timestamper if it is at the head of a VC or if the flit ahead of it in the same VC has moved to the second stage of the pipeline. A flit can also re-enter the TS stage if it fails to find either a conflict-free middle memory or a free VC at the input of the next-hop router, as will be explained later. If multiple VCs from the same input port send simultaneous requests to the timestamper, it picks a winning VC and assigns the earliest possible departure time for the output port requested by the flit in the selected VC. Let us assume that the output port requested is p. In order to find the earliest possible departure time through port p, the timestamper first computes the time the flit would leave the middle memory assuming there are no flits already stored in the middle memories that need to depart through port p. Let us denote this time as $T[p]$,

$$T[p] = \text{Current Time} + 3$$

since two pipeline stages, namely, CR+VA (Conflict resolution + VC allocation) and XB1+MM_WR (Crossbar 1 and Middle Memory Write) lie between the TS and MM_RD + XB2 (Middle Memory Read and Crossbar 2) stages in the DSB pipeline (see Figure 4). Next, we consider the case when there are flits in the middle memories destined for output port p. To handle this case, the timestamper remembers the value of the last timestamp assigned for each output port till the previous cycle. The last assigned timestamp for output port p is denoted as $LAT[p]$. As timestamps are assigned in a strictly increasing order, the assigned timestamp for output port p in the current cycle must be greater than $LAT[p]$. In other words, a flit that is currently being timestamped can depart the middle memory through output port p only after all flits that are destined for the same output port and were timestamped at an earlier cycle, depart the middle memory. This emulates the FCFS servicing scheme of OBRs. Hence, the earliest timestamp that can be assigned to a flit for output port p is given as:

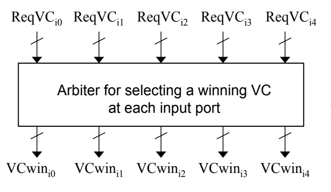
$$\text{Timestamp} = max(LAT[p] + 1, T[p]) \qquad (1)$$

If flits at more than one input port request a timestamp for output port p in the same cycle, the timestamper serves the inputs in an order of decreasing priority (either fixed or rotating). The timestamp of a flit at the highest priority input is computed as above and the remaining flits at other inputs are assigned sequentially increasing timestamps in the order of decreasing priority.

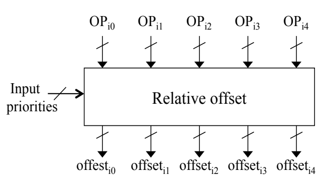
Conflict resolution (CR) and virtual channel allocation (VA) comprise the second pipeline stage of the DSB router. The CR and VA operations are carried out in parallel. The task of the CR stage is to find a conflict-free middle memory assignment for flits that were assigned timestamps in the TS stage. As mentioned earlier, there are two kinds of conflicts in shared-buffer routers – *arrival conflicts* and *departure conflicts*. Arrival conflicts are handled by assigning a different middle memory to every input port with timestamped flits. Departure conflicts are avoided by ensuring that the flits stored in the same middle memory have unique timestamps. These restrictions are enforced due to the fact that middle memories are uni-ported[3] and only one flit can be written into (via XB1) and read from (via XB2) a middle memory in a given cycle. The implementation details of the TS and CR stages are explained in the next section.

The virtual channel allocator arbitrates for free virtual channels at the input port of the next-hop router in parallel with conflict resolution. VC allocation is done only for the head flit of a packet. The VC allocator maintains two lists of VCs – a *reserved* VC pool and a *free* VC pool. VC allocation is done by picking the next free output VC from the *free* VC list of the given output port, similar to the technique used in [7]. Additionally, when output VCs are freed, their VC number

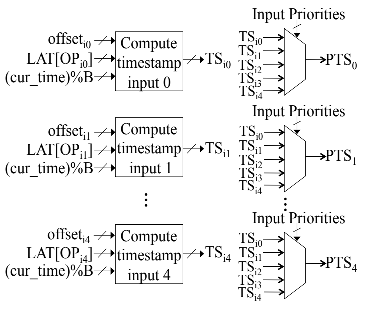
[3]Single-ported memories are power and area-efficient.


ReqVC_i0 ReqVC_i1 ReqVC_i2 ReqVC_i3 ReqVC_i4
Arbiter for selecting a winning VC at each input port
VCwin_i0 VCwin_i1 VCwin_i2 VCwin_i3 VCwin_i4

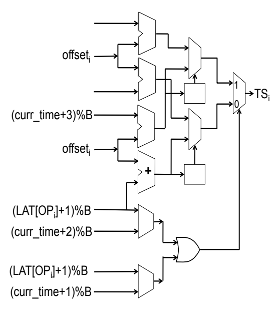
(a) Arbitration


OP_i0 OP_i1 OP_i2 OP_i3 OP_i4
Input priorities
Relative offset
offest_i0 offset_i1 offset_i2 offset_i3 offset_i4

(b) Offset generation



(c) Timestamp computation



(d) Compute timestamp block

Fig. 5. Block Diagrams of the TS stage

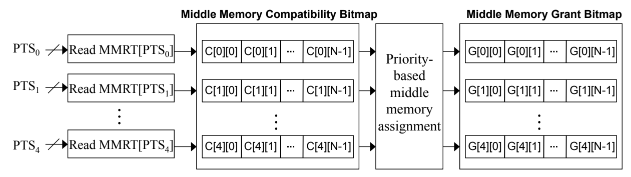

Middle Memory Compatibility Bitmap
Middle Memory Grant Bitmap
PTS_0
Read MMRT[PTS_0]
PTS_1
Read MMRT[PTS_1]
PTS_4
Read MMRT[PTS_4]
C[0][0] C[0][1] ... C[0][N-1]
C[1][0] C[1][1] ... C[1][N-1]
C[4][0] C[4][1] ... C[4][N-1]
Priority-based middle memory assignment
G[0][0] G[0][1] ... G[0][N-1]
G[1][0] G[1][1] ... G[1][N-1]
G[4][0] G[4][1] ... G[4][N-1]

Fig. 6. Block Diagram of the CR stage

is moved from the *reserved* VC list to the end of the *free* VC list. If a free VC exists and the flit is granted a middle memory, it subsequently proceeds to the third pipeline stage, where it traverses the first crossbar (XB1) and is written to its assigned middle memory (MEM_WR). If no free VC exists (all VCs belong to the reserved VC list), or if CR fails to find a conflict-free middle memory, the flit has to be re-assigned a new timestamp and it therefore re-enters the TS stage.

When the timestamp of a flit matches the current router time, the flit is read from the middle memory (MM_RD) and passes through the second crossbar (XB2) in the fourth pipeline stage. We assume that the output port information is added to every flit and stored along with it in the middle memory. Finally, in the link traversal (LT) stage, flits traverse the output links to reach the downstream router.

C. Practical implementation

In this section, we describe the implementation details of the timestamping and conflict resolution stages, which are unique to the proposed DSB architecture. It must be noted here that the proposed implementation is only one among a range of possible design implementation choices that span a spectrum of area/delay tradeoffs. We specifically focus on the implementation of a 5-ported 2D mesh router. However, our design can be extended to higher or lower radix routers.

The high level block diagrams of the logic used in the TS stage are shown in Figures 5(a)- 5(c). The five input ports are labeled from i_0 to i_4. First, as shown in Figure 5(a), when multiple VCs from the same input port send simultaneous requests to the timestamper, a winning VC is selected using a matrix arbiter. In addition to the timestamping requests, the arbiter also takes as input the size of the $free$ VC lists for the output ports requested by each of the flits in the TS stage and gives priority to body and tail flits that have already acquired a VC over head flits, if they are likely to fail in the VC allocation stage. This avoids wasted cycles resulting from re-timestamping of flits. After choosing a winning VC at each input port, the output ports requested by the flits in the selected VCs, OP_i, are used to generate timestamp offsets for each of these flits, as shown in Figure 5(b). Timestamp offsets are required to assign sequentially increasing timestamps to flits based on input priority when flits from more than one input port simultaneously request a timestamp for the same output port, as discussed in Section III-B. Internally, the offset generator block is comprised of separate sub-blocks, one for each output port. The offsets are generated on a per output port basis and its value for a flit is equal to the number of higher priority flits requesting a timestamp for the same output port. The final timestamp assigned to a flit at input i is the sum of the timestamp assigned to the highest priority flit requesting the same output port as the current flit (given by Equation 1) and the computed offset.

$$TS_i = max(LAT[OP_i] + 1, \text{current\_time + 3}) + \text{offset}_i \qquad (2)$$

In the DSB architecture, flits are stored in middle memories only after they have reserved buffering at the next-hop router. Let the total buffering at each input port be B flits. If the current time is denoted by $curr\_time$, we restrict the maximum timestamp assigned for an output port to $curr\_time + B - 1$. This is because, assigning a timestamp equal to $curr\_time + B$ means that there are B flits before the current flit (with timestamps $curr\_time$ to $curr\_time + B - 1$) that have been timestamped for the same output port and have not yet departed the router. If all timestamped flits succeed in acquiring an output VC and a conflict-free middle memory, these B flits would reserve all available buffering at the input of the next-hop router and any flit with a timestamp greater than $curr\_time + B - 1$ would fail to reserve an output VC. Hence, assigning timestamps greater than $curr\_time + B - 1$ is not necessary. This fact is used to simplify the hardware for detecting departure conflicts. From this discussion, at most B unique timestamps are assigned for an output port, which can be represented using $\lceil \log_2 B \rceil$ bits. We ensure that each middle memory has exactly B flits of buffering so that a flit with timestamp T is always stored at the T^{th} location within the middle memory. In this way, a flit with timestamp T can only have departure conflicts with flits stored at the T^{th} location of any one of the N middle memories.

With timestamps represented using $\lceil \log_2 B \rceil$ bits, the timestamp assignment has to be carried out using *modulo-B* arithmetic. Under this scheme, the current time rolls over every B clock cycles, implemented using a *mod-B* counter. The timestamps are interpreted to be relative to the current time and also roll over beyond B. Hence, if $curr\_time\%B$ has a value t, flits stored in the middle memories can have B unique timestamps between t and $(t-1)\%B$, representing times from $curr\_time$ to $curr\_time + B - 1$. If the last assigned timestamp for an output port, OP, falls behind $curr\_time\%B$ (i.e. $LAT[OP] = curr\_time\%B$), it is advanced along with the current time to ensure that the last assigned timestamp is either equal to or ahead of $curr\_time\%B$. This prevents old values of $LAT[OP]$ from appearing as future timestamps after rollover. Figure 5(d) presents the logic diagram for the timestamp computation block. When assigning a timestamp for output port OP_i, $(LAT[OP_i]+1)\%B$ is simultaneously compared to $(curr\_time+1)\%B$ and $(curr\_time+2)\%B$, and the results are *ORed* together. A 1 at the output of the *OR* gate signifies that $(LAT[OP_i]+1)\%B$ is behind $(curr\_time+3)\%B$ and vice-versa. The greater of the two times is chosen and the corresponding flit offset is added to obtain the final timestamp according to Equation 2. If the timestamp computed using Equation 2 is greater than B, it is rolled over by subtracting B from the result, as shown. In the last block of Figure 5(c), the timestamps are shuffled according to input priority, which is assumed to be a rotating priority over all inputs. In this respect, PTS_0 is the timestamp of the input with priority 0, PTS_1 is the timestamp of input with priority 1, and so on. This helps with the priority-based middle memory assignment during the CR stage. If an input does not hold a flit that needs to be timestamped, an invalid timestamp value is stored instead.

The task of the conflict resolution stage (CR) is to detect arrival and departure conflicts. To keep track of the occupancy of the middle memory buffers, we use an auxiliary data structure called the middle memory reservation table (MMRT). For N middle memories, with B flits of buffering per middle memory, the MMRT is an array of B registers, each N bits wide. The registers are indexed from 0 to $B-1$. If bit MMRT[i][j] is set, it implies that memory bank j holds a flit with timestamp i and vice versa.

Departure conflicts are resolved using the middle memory reservation table. For each timestamp that needs to be assigned a middle memory (PTS_0 ... PTS_4), the MMRT register indexed by the timestamp represents the middle memory compatibility bitmap for the timestamp. In Figure 6, the bits $C[i][0]$ to $C[i][N-1]$ represent the individual bits of the N-bit register, $MMRT[PTS_i]$. If bit $C[i][j]$ is 1, it means that middle memory j already has a flit with timestamp PTS_i and hence, has a departure conflict with any flit with this timestamp. On the other hand, if $C[i][j]$ is 0, the flit with timestamp PTS_i is compatible with middle memory j. If an input does not have a flit that needs to be timestamped, the compatibility bits for all middle memories are set to 1 (meaning incompatible).

Next, arrival conflicts are resolved in the middle memory assignment stage. The middle memories are assigned fixed priorities with memory N-1 given the highest priority and memory 0 the lowest priority. In the middle memory assignment stage, the inputs are granted the highest priority compatible middle memory in the order of decreasing input priority while ensuring that more than one input is not granted the same middle memory. Bit G[i][j] denotes the grant bit and it is set to 1 if the input with priority i has been granted middle memory j. This memory assignment scheme was specifically designed to have low middle memory miss rates when the number of middle memories is fewer than $2P-1$ (P being the number of ports) for 5-ported mesh routers. Having less than $2P-1$ middle memories is necessary to reduce the power and area of DSB routers as shown in Section V. When the number of middle memories is at least $2P-1$, memory assignment schemes with less delay can be implemented as it is much easier to find conflict-free middle memories.

The above logic distribution between the TS and CR stages was architected to even out the Fan-Out-of-4 (FO4) delays across the four stages (excluding LT) of the DSB pipeline. The FO4 calculations were carried out using the method of Logical Effort [19], and was applied to each logic block. For a 5-ported DSB router with 5 VCs per input port, 4 flits per VC (B=20 flits) and 5 middle memories with 20 flits per middle memory, the critical path delays of the TS and CR pipeline stages were estimated at 19 FO4s and 18 FO4s, respectively. A delay of less than 20 FO4 for each stage in the proposed architecture enables an aggressively-clocked high-performance implementation. In particular, assuming a FO4 delay of 15 ps for Intel's 65 nm process technology, our proposed design can be clocked at a frequency of more than 3GHz.

IV. Throughput and Latency Evaluation

A. Simulation setup

To evaluate the effectiveness of our proposed DSB router against a baseline input-buffered router (IBR) architecture with virtual channel (VC) flow-control, we implemented two corresponding cycle-accurate flit-level simulators. The baseline IBR simulator has a three-stage pipeline as described in Section II-C. The DSB simulator models the five-stage router pipeline described in Section III-B. Both simulators support k-ary 2-mesh topologies with their corresponding 5-ported routers. DOR-XY routing is used for all our simulations where packets are first routed in the X-dimension followed by the Y-dimension. We use DOR-XY because our main focus is on highlighting the improvement in performance due to the DSB router architecture, rather than the routing algorithm.

We present results for both synthetic and real traffic traces. The three synthetic traffic traces used are `uniform`, `complement` and `tornado` traffic, shown in Table I. These three traces represent a mixture of benign and adversarial traffic patterns. The ideal saturation throughputs that can be achieved for these three traffic patterns using DOR-XY (based on channel load analysis) are also shown in Table I. All throughput results presented subsequently are normalized to the ideal saturation throughput for the given traffic pattern. An 8×8 mesh topology is used for our simulations with synthetic traffic. Multi-flit packets composed of four 32-bit flits are injected into the network and the performance metric considered is the average packet latency under different traffic loads (packet injection rates). The latency of a packet is measured as the difference between the time the header flit is injected into the network and the time the tail flit is ejected at the destination router. The simulations are carried out for a duration of 1 million cycles and a warm-up period of ten thousand cycles is used to stabilize average queue lengths before performance metrics are monitored.

In addition to the synthetic traces, we also compare the performance of the two router architectures on eight traffic traces from the SPLASH-2 benchmark suite [17]. The traces

TABLE II
NOMENCLATURE OF THE ROUTER MICROARCHITECTURES COMPARED

Config.	Input buffers (per port)		Middle memory buffers		Total buffers
	#VCs	#Flits/VC	#Middle Memories (MM)	Flits/MM	
IBR200	8	5	-	-	200
DSB200	5	4	5	20	200
DSB300	5	4	10	20	300

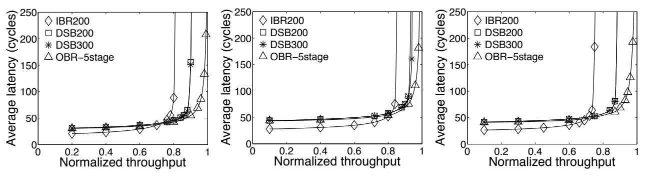
used are for a 49-node shared memory CMP [8] arranged as a 7×7 mesh. The SPLASH-2 traces were gathered by running the corresponding benchmarks with 49 threads[4] on Bochs [9], a multiprocessor simulator with an embedded Linux 2.4 kernel. The memory trace was captured and fed to a memory system simulator that models the classic MSI (Modified, Shared, Invalid) directory-based cache coherence protocol, with the home directory nodes statically assigned based on the least significant bits of the tag, distributed across all processors in the chip. Each processor node has a two-level cache (2MB L2 cache per node) that interfaces with a network router and 4GB off-chip main memory. Access latency to the L2 cache is derived from CACTI to be six cycles, whereas off-chip main memory access delay is assumed to be 200 cycles. Simulations with SPLASH-2 traces are run for the entire duration of the trace, typically in the range of tens of millions of cycles, which is different for each trace.

B. Performance of the DSB router on synthetic traces

The nomenclature of IBR and DSB configurations referred to in this section is presented in Table II, along with details of the buffer distribution in each configuration. For the DSB architecture, the number of flits per middle memory is always equal to the number of input buffers to simplify departure conflict resolution, as discussed in Section III-C. Theoretically, a 5-ported DSB router needs 9 middle memories to avoid all conflicts in the worst case, i.e., when all possible arrival and departure conflicts occur simultaneously. However, keeping the power overhead in mind, we evaluate a DSB configuration with only 5 middle memories (DSB200) and compare its performance to a configuration with 10 middle memories (DSB300).

In addition to the configurations shown in Table II, we also implemented an OBR simulator with a very large number of buffers (10,000 flit buffers) at each output port, emulating infinite output buffer capacities. Redundant pipeline stages are introduced in the OBR simulator totaling a pipeline depth of five, to ensure the same pipeline length as our DSB router for fair comparisons. This helped us to compare the performance of DSB configurations with an OBR with same number of pipeline stages, but which behaves ideally and incurs no delay due to switch arbitration and buffer capacity limitations. We refer to this OBR configuration as OBR-5stage.

We first compare the performance of IBR200, DSB200, DSB300 and OBR-5stage. IBR200 and DSB200 have the same number of buffers (200 flits). DSB300 has the same number of input buffers but double the number of middle memories compared to DSB200. The average packet latency curves for different injected loads are shown in Figure 7 for the three synthetic traffic patterns and the saturation throughput values are presented in Table III. The saturation throughput is assumed to be the injection rate at which the average packet latency is three times the zero-load latency. Table III shows that with an aggregate buffering of 200 flits, DSB200 outperforms IBR200 by 11.25%, 9.5% and 18.5% on uniform, complement and tornado traffic, respectively, in terms of saturation throughput. Although IBR200 has a slightly lower latency than DSB200 under low loads due to the shorter router pipeline, the higher saturation throughput of DSB200 gives it a definite edge under moderate to high loads. It must be noted here that during the course of an application running on a CMP, there may be transient periods of high traffic or localized traffic hotspots during which parts of the network are driven close to (or past) saturation. A router with higher saturation throughput can minimize the occurrence of these transient hotspots and provide tremendous latency savings during these periods, which can far outweigh the slightly higher latency under low loads. This is more clearly depicted in the SPLASH-2 results presented in Section IV-C.



(a) Uniform (b) Complement (c) Tornado

Fig. 7. Performance comparison of different router architectures.

TABLE III
NORMALIZED SATURATION THROUGHPUT COMPARISON FOR DIFFERENT NOC ROUTER MICROARCHITECTURAL CONFIGURATIONS

Traffic pattern	IBR200	DSB200	DSB300	OBR-5stage
Uniform	80%	89%	89%	98%
Complement	85%	93%	94%	97%
Tornado	75%	89%	89%	97%

The performance of DSB200 is nearly identical to DSB300, with negligible difference in saturation throughputs for all three traffic patterns. This is because the probability of more than five arrival and departure conflicts occurring simultaneously is very low. We observe in our experiments that even under very high injection loads, less than 0.3% of the flits failed to find a conflict-free middle memory in the worst case over all traffic patterns. Hence, it can be concluded that although theoretically nine middle memories are needed to resolve all conflicts, in practice, five middle memories result in very little degradation in throughput. The reason fewer middle memories are attractive is because of the significant power and area savings that can be achieved by using smaller crossbars and fewer buffers.

The saturation throughput of DSB200 is also quite close to that of OBR-5stage. For uniform, tornado and complement traffic, the saturation throughput of the DSB200 architecture is within 9%, 4% and 8%, respectively, of the throughput of OBR-5stage. The slightly lower saturation throughput of DSB routers is a result of having far fewer buffers compared to OBR-5stage's infinite buffering.

[4]Two of the eight traces, `fft` and `radix` could not be run with 49 threads. They were run with 64 threads instead. The memory traces of the first 49 threads were captured and the addresses were mapped onto a 49 node system.

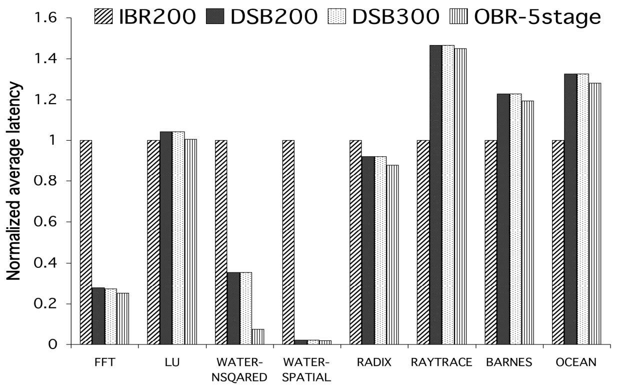


Fig. 8. Network latency for SPLASH-2 benchmarks.

C. Performance of DSB on real traffic traces

In this section, we present simulation results using the eight benchmark traces from the SPLASH-2 suite [17]. For uniform, tornado, and complement traffic, the traffic matrix Λ is assumed to be fixed and stationary. As discussed in Section II-A, using idealized load analysis, the ideal saturation throughput can be computed based on just Λ in these cases. However, in the case of real traffic traces, like the SPLASH-2 traces, the traffic pattern is space- and time-varying. Therefore, the notion of ideal saturation throughput can neither be clearly defined nor easily determined. Instead, we compare the average packet latencies over the entire trace duration using our cycle-accurate flit-level simulators.

Figure 8 shows the latency results for IBR200, DSB200, DSB300 and OBR-5stage configurations normalized to the average packet latency of IBR200. The first observation to be made here is that the average packet latency of the DSB configurations is comparable to OBR-5stage for seven of the eight SPLASH-2 benchmarks evaluated. For the `water-nsquared` trace, OBR-5stage has a lower latency than DSB200 (and DSB300) because this trace has traffic hot spots where packets are injected at a rate where both OBR and DSB routers get saturated. In such a situation, the large number of output buffers available in OBR-5stage help in attaining a lower average latency. In most traces, the ability of a DSB router to closely match an OBR in terms of latency illustrates its capability to emulate OBRs, even with limited buffering.

Next, we observe that DSB200 outperforms IBR200 on `fft`, `water-nsquared`, `water-spatial` and `radix` traces by 72%, 64.5%, 97.5% and 8%, respectively. For the three traces where the performance improvement is over 50%, i.e., `fft`, `water-nsquared` and `water-spatial`, IBR200 gets saturated during portions of all three traces while DSB200 saturates only in the case of the `water-nsquared` trace. So, it can be inferred from these results that a relatively small increase in saturation throughput translates into tremendous reductions in packet latency for applications that demand high bandwidth. However, IBR200 has 32%, 18% and 24% lower latency for `raytrace`, `barnes` and `ocean` benchmarks. This is because these traces have negligible output port contention and the higher delay of DSB routers is due to their longer pipeline depth of 5 as compared to 3 for IBR200. This is proved by the fact that even OBR-5stage, which has no extra delay introduced as a result of switch contention, has higher average packet latency than IBR200. For the SPLASH-2 benchmarks with high output port contention (`fft`, `water-nsquared`, `water-spatial` and `radix`), on an average, DSB200 has 60% lower latency than IBR200. Hence, for applications that demand high throughput and drive the network towards saturation or close to saturation, DSB routers are clearly superior to IBRs.

Lastly, there is negligible difference in performance between DSB200 with 5 middle memories and DSB300 with 10 middle memories. This further proves that even with real application traces, more than five simultaneous arrival and departure conflicts occur very rarely. Hence, 5 middle memories are sufficient to achieve comparable performance to a 10 middle memory configuration, but at a significantly lower power and area overhead, as will be discussed in the next section.

V. Power and Area Evaluation

Table IV compares the power and area of IBR and DSB router microarchitectures with the same aggregate buffering. We use the power and area models in Orion 2.0 [5], [21] for our analysis. The models use parameters from TSMC 65nm process libraries and include both dynamic and leakage power components. The operational frequency used is 3GHz at 1.2V. A typical flit arrival probability of 0.3 is assumed at each input port. SRAMs are used as input and middle memory buffers and low threshold voltage transistors are used to ensure low delays for 3GHz operation. The VC allocator is configured to simply select a free VC from a free VC list as described in Section III-B in both DSB and IBR architectures while a matrix arbiter is used for switch arbitration in IBRs. The power and area of the arbitration logic of the DSB router were extrapolated from the power and area numbers of the arbitration logic of an IBR using the number of 2-input gates required to implement the logic in the two architectures.

As shown in Table IV, DSB200 consumes 35% more power and occupies 58% more area than the corresponding IBR200 router. The higher power and area of the DSB router is due to the presence of an extra crossbar and a more complex arbitration scheme (involving timestamping and conflict resolution) compared to the switch arbiter in an IBR.

Although the increased power cost per router for a DSB router is substantial when compared to an IBR, the overall power increase for a NoC application is often less. In Table IV, the power penalty per tile (processor + router) of using a DSB router is presented for three different scenarios where the NoC consumes 10%, 15% and 20% of the total tile power. Even for applications where the router consumes as high as 20% of the tile power, the power per tile with a DSB200 router is only 7% higher than the tile power with IBR200. On the other hand, if the router consumes only 10% of the tile power, the power per tile with a DSB200 router is just 3.5% higher than tile power with IBR200. We believe that the increased power cost is justified for applications that demand high bandwidth and exhibit high contention since latency reductions of more than 60% can be achieved using DSB routers. As with power, even in the case of area, the increase in area of the entire tile as a result of using a DSB router in place of an IBR is again

TABLE IV
ROUTER POWER AND AREA COMPARISON

Router Config.	Power (mW)	Area (mm^2)	Router Config.	Power (mW)	Area (mm^2)	Power penalty			
						Per router	Per tile		
							NOC power = 10%	NOC power = 15%	NOC power = 20%
IBR200	240	0.19	DSB200	324	0.3	1.35	1.035	1.052	1.07

TABLE V
COMPARISON OF DSB ROUTERS WITH 5 AND 10 MIDDLE MEMORIES

Router Configuration	Power (mW)	Area (mm^2)
DSB200	324	0.3
DSB300	496	0.5

very low since the router area is only a small portion of the total tile area.

As discussed earlier, the DSB200 configuration with 5 middle memories and DSB300 configuration with 10 middle memories have similar performance. However, DSB200 consumes 35% less power and occupies 40% less area compared to DSB300 (Table V). The power and area savings are achieved by using fewer buffers and two 5×5 crossbars in DSB200 instead of 5×10 and 10×5 crossbars used in DSB300.

VI. RELATED WORK

Sophisticated extensions to input-buffered router microarchitectures have been proposed for improving throughput, latency and power. For throughput, techniques like flit-reservation flow-control [13], variable allocation of virtual channels [12] and express virtual channels [8] have been proposed. As these designs are input-buffered, they are only able to multiplex arriving packets from their input ports across the crossbar switch, unlike our proposed DSB architecture which can shuffle incoming packet flows from all input ports onto the middle memories and then onto the crossbar switch. The proposed DSB architecture offers better opportunities for packet multiplexing and improved packet flow, which helps in mimicking the high throughput and predictable delay characteristics of output-buffered routers. There have been several input-buffered router proposals that target network latency, making single-cycle routers feasible, such as speculative allocation [10], [11], [14] and route lookaheads [3], [7]. For power savings, techniques such as row-column separation [6] and segmentation [22] of crossbars and straight-through buffers [22] have been proposed. These latency and power optimization techniques are orthogonal to our proposal as they do not target throughput. Some of these techniques can be applied to the DSB router as well to reduce latency and energy consumption.

As already mentioned, DSB routers [4], [15], which can emulate an output-buffered router without router speedup, have been successfully used in Internet routing. Stunkel et. al [18] proposed the IBM colony router which is a customized architecture for off-chip interconnection networks with large central buffers and three crossbars. Although this architecture is similar to that of DSB routers, it does not use timestamping of flits for OBR emulation. Instead, packets potentially incur large de-serialization and serialization latencies to support wide SRAM accesses.

VII. CONCLUSIONS

In this paper, we proposed a distributed-shared-buffer (DSB) router for on-chip networks. DSB routers have been successfully used in Internet routers to emulate the ideal throughput of output-buffered routers, but porting them to on-chip networks with more stringent constraints presents tough challenges. The proposed DSB router achieves up to 19% higher saturation throughput than input-buffered routers (IBRs) and up to 94% of the ideal saturation throughput for synthetic traffic patterns. The higher saturation throughput translates to large reductions in network latency with SPLASH-2 benchmarks. For SPLASH-2 applications which exhibit high contention and demand high communication bandwidth, DSB routers on an average have 60% lower network latencies than IBRs.

ACKNOWLEDGMENT

This work was supported in part by NSF (grant no. CCF 0702341).

REFERENCES

[1] W. J. Dally. Virtual-channel flow control. In *ISCA*, May 1990.
[2] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004.
[3] P. Gratz *et al.* Implementation and evaluation of on-chip network architectures. In *ICCD*, Oct. 2006.
[4] S. Iyer, R. Zhang, and N. McKeown. Routers with a single stage of buffering. In *ACM SIGCOMM*, September 2002.
[5] A. Kahng *et al.* Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *DATE*, April 2009.
[6] J. Kim *et al.* A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *ISCA*, June 2006.
[7] A. Kumar *et al.* A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *ICCD*, Oct. 2007.
[8] A. Kumar *et al.* Express virtual channels: Towards the ideal interconnection fabric. In *ISCA*, June 2007.
[9] K. P. Lawton. Bochs: A portable PC emulator for Unix/X. *Linux J.*, 1996(29):7, 1996.
[10] S. S. Mukherjee *et al.* The Alpha 21364 network architecture. *IEEE Micro*, 22(1):26–35, Jan./Feb. 2002.
[11] R. Mullins *et al.* Low-latency virtual-channel routers for on-chip networks. In *ISCA*, June 2004.
[12] C. A. Nicopoulos *et al.* ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *MICRO*, Dec. 2006.
[13] L-S. Peh and W. J. Dally. Flit-reservation flow control. In *HPCA*, Jan. 2000.
[14] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA*, Jan. 2001.
[15] A. Prakash, A. Aziz, and V. Ramachandran. Randomized parallel schedulers for switch-memory-switch routers: Analysis and numerical studies. In *IEEE INFOCOM*, March 2004.
[16] V. Soteriou *et al.* A high-throughput distributed shared-buffer noc router. *Computer Architecture Letters*, 8(1), 2009.
[17] SPLASH-2. http://www-flash.stanford.edu/apps/SPLASH/.
[18] C. B. Stunkel *et al.* A new switch chip for IBM RS/6000 SP systems. In *ICS*, 1999.
[19] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, 1999.
[20] Brian Towles and William J. Dally. Worst-case traffic for oblivious routing functions. In *SPAA*, pages 1–8, Aug. 2002.
[21] H.-S. Wang *et al.* Orion: A power-performance simulator for interconnection networks. In *MICRO*, Nov. 2002.
[22] H.-S. Wang *et al.* Power-driven design of router microarchitectures in on-chip networks. In *MICRO*, Nov. 2003.