Path-Diverse Inorder Routing

Mieszko Lis, Myong Hyon Cho, Keun Sup Shim, and Srinivas Devadas
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{mieszko, mhcho, ksshim, devadas}@mit.edu

ABSTRACT

We present Path-Diverse In-Order Routing (PDIOR), an oblivious routing method which guarantees network-level inorder delivery for multi-path routing. Based on Exclusive Dynamic Virtual Channel Allocation (EDVCA), which allows single-path efficient in-order delivery with dynamic virtual channel allocation, PDIOR extends the same guarantees to routing schemes where each flow may be routed via more than one path. As with EDVCA, PDIOR avoids the overheads inherent in reordering packets at the destination core, and requires only minor, inexpensive changes to traditional oblivious router architectures: for example, an implementation of PDIOR on 8×8 mesh network with 4 VCs per port requires 492 bytes of memory per node, while inorder packet delivery in a comparable conventional network may requires tens to hundreds of kilobytes of reorder buffer memory at each node.

1. INTRODUCTION

Inorder packet delivery in a network is a widely assumed basis for a wide range of application protocols such as file transfer protocols and optimized cache coherence protocols (e.g., [6, 7]); for example, Hennessy & Patterson begin the description of their cache coherence protocol with "first, we assume that the network provides point-to-point inorder delivery of messages" [6, p. E–7]. Implementations of direct-communication computation models such as stream computing (e.g., StreamIt [15]) also require that packets be delivered in the order they were sent, as do explicit message-passing applications. Indeed, inorder delivery is so widely taken for granted that it is often not specifically addressed.

Although some routing algorithms naturally deliver packets in order, they offer limited efficiency. Basic dimension-order routing (DOR) without virtual channels, an approach popular in network-on-chip (NoC) designs, always preserves packet order because all packets follow the same path and are stored in the same buffers. Because packets from different flows are buffered in the same queues, however, a single ill-behaved flow can overwhelm other flows and effectively block them even if they are destined for a different egress port, a phenomenon known as head-of-line blocking.

Unfortunately, popular solutions to these shortcomings either sacrifice inorder delivery for improved performance or have limited application. "Path-diverse" routing protocols (e.g., [13, 16, 11, 12]) alleviate throughput problems by routing each flow via more than one path, but give up inorder delivery, since packets on the different paths may experience different congestion and travel times. Static route assignment techniques like WOT [4] or BSOR [8] can optimize flow-to-route assignment to limit congestion, but rely on precomputing routes off-line and configuring the network before the application starts. This requires fairly accurate a priori knowledge of the application's traffic patterns, a reasonable assumption for fixed-application chips but an unrealistic requirement for general-purpose NoCs. Using multiple dynamically allocated virtual channels (VCs) on each link, a popular way to ameliorate head-of-line blocking, allows packets from the same flow to be buffered in multiple VCs on a given link, and, in effect, creates multiple virtual paths for each flow, compromising inorder guarantees. Allocating VCs statically [14] can minimize head-of-line blocking and maximize throughput, but again relies on off-line precomputation and as such is not generally applicable.

In such algorithms, inorder delivery can be accomplished by resorting to packet reordering: each packet is tagged with a sequential serial number, and any packets that arrive out of order are stored in a reorder buffer at the destination node until all of their predecessors have been received. This induces significant hardware cost, as the reorder buffers at each node must be quite large to ensure that all out-of-order packets can be stored: in simulations using DOR and O1TURN on a 4-VC system, we found that, depending on the traffic pattern, up to between 25% and 75% of packets arrived out of order, and the reorder buffers needed to hold up to 69 packets. The high percentage of packets received out of order indicates that the reorder buffer and reordering logic must operate at close to line rate, effectively excluding any software-level solution. Since one such buffer may have to be implemented for each flow arriving at a given destination, and the efficiency demands would require very fast storage, the cost of reorder buffer space alone in a store-and-reorder scheme would be significant.

In this paper, we describe efficient *network-level* in-order delivery in multi-path oblivious algorithms like O1TURN [13] with multiple dynamically allocated VCs. By limiting the number of outstanding packets on each path, PDIOR guarantees inorder delivery while avoiding deadlock, reorder buffers, and retransmission logic. In addition, PDIOR applied to O1TURN has 6% better performance on average compared to baseline (out-of-order) O1TURN on synthetic and application loads due to reduced head-of-line blocking and improved load balancing efficiency.

PDIOR requires only minor, inexpensive changes to traditional oblivious router architectures. Unlike reorder buffers, which may require on the order of 10–100 KB *per destination core* to keep up with line rate and grow with packet size, the additional memory required by PDIOR (492 bytes for an 8×8 mesh) is insignificant, and is independent of dynamic traffic properties.

While our motivation in this paper is inorder packet delivery in NoCs and we focus on applying PDIOR (Section 3) to O1TURN routing on a mesh, PDIOR is independent of network topology and route selection, and can be applied to other oblivious path-diverse routing algorithms (e.g., Valiant [16], ROMM [11, 12], or adaptive routing schemes like turn methods [5] or odd-even routing [2].

In Section 3 we describe how PDIOR implements inorder packet delivery. Next, Section 4 details implementation differences relative to a baseline oblivious virtual-channel router design. Section 5 offers performance analysis via extensive cycle-accurate simulation with synthetic as well as application traffic patterns, and Section 6 concludes the paper.

2. RELATED WORK

Few routing scheme designs explicitly address out-of-order packet delivery. Within the Network-on-Chip (NoC) context, DOR routing without virtual channels is naturally ordered, since all pack-

ets between a specific source and destination travel along the same path and are buffered in the same queues, but the ordering guarantee breaks down with multiple VCs. ROMM [11], Valiant [16] and O1TURN [13] may all deliver packets out of order. Static route/VC assignment schemes, such as Weighted Ordered Toggle [4] and BSOR [8, 14], can guarantee ordering but rely on off-line route/VC assignment and require knowledge of traffic patterns for efficiency.

EDVCA [9] and Flow-Aware Allocation (FAA) [1] are similar dynamic VC allocation schemes that can work with any routing algorithm. While EDVCA focuses on inorder guarantees (and defines flows as source-destination pairs) and FAA focuses on ameliorating head-of-line blocking (and defines flows by destination only), both can guarantee inorder packet delivery for single-path routing algorithms; neither, however, conserves ordering in a multi-path routing scheme where each flow can travel via several distinct paths with potentially different travel times.

Murali et al [10] describe a multi-path inorder scheme where sequentially numbered packets belonging to a given flow are delayed at switches where distinct paths used by the same flow join (or cross), their scheme also relies on a static assignment of flows to links; moreover, their reordering method contemplates only packets within one flow and either does not consider the possibility of deadlock when separate flows block each other or makes the unrealistic assumption of a private virtual channel for each flow.

3. PATH-DIVERSE INORDER ROUTING

PDIOR implements inorder packet delivery guarantee for multi-path routing algorithms. We start by outlining EDVCA, which ensures that packets are delivered in order along a single path (i.e., sequence of nodes), and then describe how PDIOR extends this guarantee to flows which may travel along multiple paths.

We describe and evaluate our scheme as implemented on top of O1TURN [13] routing in a 2D mesh. O1TURN randomly chooses between XY and YX routing for each packet, thus on average sending 50% of traffic along each path; to avoid deadlock, separate virtual channel sets are used for the XY and YX paths.

3.1 Single-path ordering

Even single-path routing schemes (e.g., DOR) can deliver packets out-of-order when routers have multiple dynamically allocated virtual channels. EDVCA [9] overcomes this limitation and guarantees inorder delivery in single-path algorithms by ensuring that packets from each flow are buffered in at most one VC per node, and thus effectively travel through only one *sequence of VCs* at any one time even though the sequence of VCs may change over time.

To achieve this, EDVCA alters the VC assignment logic and the related credit update mechanism. When allocating a next-hop VC to a packet from a flow f, the following principles apply:

- if no next-hop VC contains packets from f, assign the packet to any available VC; if no VCs are available, stall the packet and allocate again in the next cycle (emulates dynamic VCA)
- if some next-hop VC v already contains packets from f, and v is available, assign the packet to v; if v is not available, stall the packet and try to allocate again in the next cycle.

In a traditional VC router, each router keeps track of the number of free slots (credits) in every next-hop VC queue, only considering a packet's flits for crossbar traversal when the credit counter for the relevant remote VC queue is positive; the next-hop router sends back per-VC credit updates when flits are forwarded from its ingress queues. EDVCA adds a Flow Assignment Table (FAT), which ensures that a flow is assigned to at most one next-hop VC at any given time. The FAT entry for each flow lists the currently assigned VC (if any), and the number of flits from that flow remaining in that VC; to keep the FAT up to date, the credit updates from the next-hop routers include the flow IDs of the departed flits. During VC allocation, EDVCA queries the FAT in parallel with standard dynamic VCA, and overrides its result if the flow already has flits in the next-hop VC:

- choose the next available VC v according to a dynamic VC scheduling algorithm;
- in parallel, query the FAT entry for flow f, FAT$[f]$;
- if FAT$[f]$ names a VC and #flits > 0, assign flow f to the VC in FAT$[f]$;
- otherwise, assign f to v and set FAT$[f] \leftarrow$ (VC$=v$, #flits$=0$).

3.2 Multi-path ordering

Much like EDVCA ensures that packets from one flow travel via at most one *sequence of VCs* at any point in time, PDIOR ensures that at any snapshot packets flow through only one *sequence of nodes* even though the sequence may be different at different times. Under PDIOR, the source node chooses one of the available routes, and sends some number of packets (drawn from a random distribution with some expected value N) along that route. The last contiguous packet sent on that route is tagged with a "switch" flag indicating that the next packet will be sent along a different route; upon receiving a "switch" packet, the destination router sends an acknowledgement (ACK) packet back to the source. Finally, when the ACK is received, the source node selects a different path and sends another sequence of packets. This way, the flow only travels along one path at any given instant, allowing inorder delivery; at the same time, when examined over an extended period of time, the flow travels over both paths and benefits from the congestion-reducing effects of path diversity.[1]

Naturally, since transmission stalls while the source node waits for the ACK packet, efficiency depends on the number of contiguous packets sent on a single path (the N above). If the N is too small, the period when packets are being transmitted (the *on_time*) will be small compared to the time spent waiting for the ACK (the *off_time*) and overall throughput will be poor; if the N is too large, on the other hand, PDIOR will spend long stretches in either XY or YX mode, and consequently may suffer from the same limitations as DOR. PDIOR addresses this by "learning" on the fly a value of N which balances the two extremes,[2] as described in Section 3.4 below. Like O1TURN, PDIOR uses two sets of virtual channels to achieve deadlock freedom: one set is used for traffic on the XY path, and the other for traffic on the YX path. PDIOR also uses EDVCA for VC allocation, so packets can flow through only one sequence of nodes and only one sequence of VCs to preserve the packet order.

3.3 Route control

The additional state required by PDIOR is contained in a Route Control Table (RCT) in each node at which traffic originates. For each flow that starts at the node, the RCT stores:

- the *current-route* value (XY or YX),
- the *current-N* value of the average sequence length,
- a *waiting-for-ACK* flag indicating whether the flow is transmitting or waiting for an ACK packet,
- the *on_timestamp* (in cycles),

[1] Unlike Weighted Ordered Toggle [4], which chooses either XY or YX routing for each flow *before runtime* and therefore offers no path diversity for any one flow, PDIOR changes the routing *on the fly* and therefore offers true run-time path diversity.

[2] PDIOR remains an oblivious algorithm, since, unlike an adaptive routing algorithm, it does not reroute packets in response to regional or global congestion statistics.

- the *off_timestamp* (in cycles).

Packets on a given flow f may be transmitted by the source node whenever it is not waiting on an ACK for f:

- if RCT[f] has *waiting-for-ACK* set, stall;
- otherwise $r \leftarrow [0 \ldots 1)$ uniformly at random, and
- if $r < \frac{1}{current-N}$,
  - set the packet's *switch* flag,
  - send the packet on *current-route* in RCT[f],
  - set *current-route* $\leftarrow$ inverse of *current-route*,
  - set *waiting-for-ACK*, and
  - set *off_timestamp* $\leftarrow$ current time.
- otherwise,
  - clear the packet's *switch* flag, and
  - send the packet along *current-route* in RCT[f].

The destination node receives packets as it normally would under O1TURN, except that for any packet with the *switch* flag, say on flow f, it also generates an ACK packet for flow f and sends it to f's source along any path. Finally, when an ACK packet for flow f is received at the source node,

- clear *waiting-for-ACK* in RCT[f],
- set *on_timestamp* $\leftarrow$ current time, and
- adjust *current-N* (see Section 3.4 below).

Flow f is then free to continue transmission.

3.4 Switching frequency control

To find a good sequence length N, PDIOR aims to keep the ratio of the *off_time* (when packets are *not* being transmitted) to the *on_time* (when packets *are* being transmitted) within a heuristically determined range. The *on_time* can be computed locally at the source node, and the *off_time* is the number of cycles between when the switch packet is transmitted and the ACK packet received. N is then adjusted as follows:

- if $off\_time > on\_time/l_{th}$, N is multiplied by $2^{\lceil log_2(l_{th} \cdot off\_time/on\_time) \rceil}$
- if $off\_time < on\_time/h_{th}$, N is divided by $2^{\lceil log_2(on\_time/(h_{th} \cdot off\_time)) \rceil}$

The parameter l_{th} gives the lower boundary of "duty cycle" D, that is $on\_time/(on\_time + off\_time)$. In PDIOR, throughput during *on_time* is scaled down roughly by D because packets are not injected in *off_time*; hence PDIOR cannot outperform single-path EDVCA if throughput during *on_time* is cannot exceed the throughput of single-path EDVCA after being scaled down by D. We address this by requiring a minimum duty cycle: for example, in an 8-by-8 mesh, we set the minimum duty cycle to be $1/1.5$, with $l_{th} = 2$, because the worst-case throughput of O1TURN routing is about 150% greater than the worst-case throughput of DOR assuming that throughput during *on_time* is roughly the same as O1TURN.

3.5 Effects

PDIOR guarantees network-level inorder packet delivery in a multi-VC network, while maintaining the congestion-robustness and fault-tolerance advantages of path-diverse routing and reduction in head-of-line blocking due to EDVCA on multiple VCs. Although the need to wait for acknowledgement before switching paths slightly lowers the possible throughput, automatically adjusting the contiguous packet sequence length N as described in Section 3.4 ensures that this *off_time* is low compared to the *on_time* during which packets are actually transmitted, and, as shown in Section 5, often exploits the benefits gained from path diversity.

While in this paper we focus on a two-path version of PDIOR based on O1TURN in a 2D mesh geometry, the technique can be used in any connection geometry, and can be applied to any path-diverse algorithm where the route can be selected at the source (i.e., not along the way): instead of inverting the route choice (XY vs. YX), one would randomly select among the available routes.

4. IMPLEMENTATION COST

4.1 Conventional Inorder Network

Ensuring inorder packet delivery in a fast on-chip network incurs some additional cost in all but the most basic routing schemes (e.g., single-VC DOR); for example, a store-and-reorder scheme with good performance would require significant buffer space at the destination as discussed in Section 1.

If the size of reorder buffer is not enough to hold all out-of-order packets until they can be reordered, either end-to-end flow control or retransmission is required to ensure that the destination buffers do not overflow, which requires additional memory space at the source and degrades throughput due to additional protocols.

Instead of dedicated reorder buffers at network nodes, out-of-order packets may be stored in the main memory of processing elements. Even so, any out-of-order packets must be removed from the network at line rate in order to prevent deadlock, which imposes the severe requirement that the memory be fast enough to keep up; equipping a processing element with enough such memory to satisfy both the reorder buffers and the applications it runs can be prohibitively expensive.

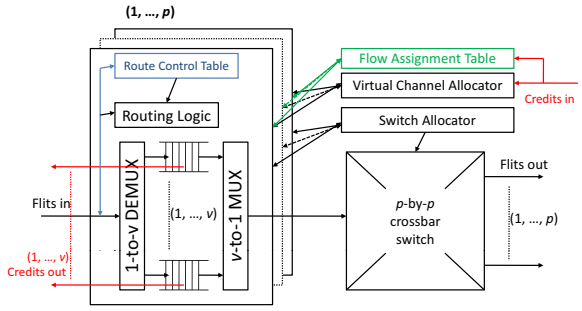
Thus, conventional implementations of inorder packet delivery either require a large amount of fast memory at each node, or significantly degrade the performance. In comparison, as shown below, PDIOR requires only a modicum of additional hardware.

4.2 PDIOR

As PDIOR uses EDVCA on each path, each node needs a Flow Assignment Table (FAT), which maps each flow ID to a remote VC assignment (two bits for a 4-VC system) and a flit count (three bits for eight-flit queues per VC); the table key is a flow ID, which, assuming a flow for each source-destination pair in an 8×8 system, might be twelve bits. While at first blush it might appear that in a system with many flows the FAT might have to be quite large—e.g., 4096 entries in an 8×8 mesh—observe that only a much smaller subset of flows will ever pass through any single node. In O1TURN-based PDIOR, this is limited to flows where either the source or the destination node is on the same row or column as the relevant transit node: in an $N \times N$ mesh the maximum number of flows that pass through a single node is $N^2(2N-1)/4$ with even N or $(2N-1)(N^2-1)/4$ with odd N, for a total of 240 bytes in an 8×8 mesh with 4 eight-flit VCs.

PDIOR also needs a Route Control Table (RCT) at each *source* node (see Section 3.3). Although this table is nominally addressed by the flow ID, such IDs generally encode the source and destination node IDs; since the RCT lives at the source node, its node ID can be ignored, and in an 8×8 system with a flow between each source/destination pair, each source node would need a 63-entry table. Each entry in the table requires a bit for the route direction (XY or YX), a bit for the ACK flag, several (say 15) bits for the sequence length and the off-timestamp (which can be relative), for a total of 32 bits (the on-timestamp can be omitted as the on-time can be approximated from N).

In all, PDIOR requires a total of 492 bytes per node (240 bytes for the FAT, and 252 bytes for the RCT). In comparison, a reordering scheme might need to buffer more than 50 packets with their timestamps (see Section 1) *for each flow arriving at a node*, which translates to 6,500 bytes for 128-byte packets with two-byte times-



1: Router architecture with PDIOR support, with significant deltas from the traditional virtual-channel router highlighted in green (FAT) and blue (RCT).

tamps when only one flow arrives at each node, and potentially hundreds of kilobytes if a node receives multiple flows.

A path-diverse O1TURN router already randomly selects a route for every packet at the source with 50% probability. PDIOR adds an overhead of an additional RCT lookup before the random selection (because N must be retrieved from the RCT), and requires more random bits per query (because the probability varies with N); this, however, is easily pipelined and does not limit throughput. A small amount of additional logic is required for ACK handling and adjustment of the average sequence length N: arriving head flits must be examined for the *switch* bit, and an ACK packet generated for each *switch* packet; in addition, arriving ACK packets must be discarded and their arrival times used to recompute N in the RCT.

Finally, PDIOR causes a small traffic overhead. The additional *switch* flag required in each head flit will not affect flit size in most designs, as the other information (flow ID and length) need not take up the entire bit width of the flit, so the overhead is caused entirely by the ACK packets. These packets themselves need not carry any data and so can be limited to just one flit; even with small packets and relatively small N, this does not appreciably increase the amount of traffic on the network.

These small overheads compare favorably with the resources and logic required to implement a typical store-and-reorder scheme for inorder delivery. Unlike reorder buffers, the additional table memories do not grow with maximum packet size, and the additional VC allocation, credit update, and route control logic are much simpler than the logic needed to reorder, acknowledge, and possibly store and retransmit packets.

5. EXPERIMENTAL RESULTS

We have evaluated the performance of PDIOR via extensive simulation on synthetic benchmarks as well as a load profile obtained from a parallel implementation of an H.264 video decoder, and report the results below.

5.1 Experimental setup

We compared throughput and latency of O1TURN-based PDIOR to dynamic-VCA DOR, dynamic-VCA O1TURN (O1TURN-OoO), and also DOR-based single-path EDVCA (EDVCA). While the baseline DOR and O1TURN with dynamic VCA do not guarantee inorder packet delivery, comparing against them shows that inorder delivery can be implemented without a performance penalty.

For our experiments, we used an in-house cycle-accurate NoC simulator, which implements an ingress-queued standard virtual-channel router [3]. To mitigate crossbar cost with routing schemes that require multiple VCs to avoid deadlock, a VC output multiplexer chooses a subset of the VCs at each ingress and presents the chosen subset for switch allocation. To avoid unfairness effects resulting from a particular combination of round-robin strategy and packet arrival rate, VCs in switch and VC allocation are considered in random order and greedily matched. To estimate the impact of out-of-order packets, we implemented a store-and-reorder strategy, although reorder buffer sizes are not limited and so retransmission is never necessary.

Table 1 summarizes the configurations used for the experiments that are shown here. We repeated the experiments on 2 VCs and an 8×8 crossbar with 4 ports from the processor to the switch; as the results as they have the same flavor as those presented, we omit them for brevity.

5.2 Throughput analysis

It is not surprising that multi-path routing can significantly outperform single-path routing by reducing link congestion and load-balancing traffic; for example, on bit-reverse and transpose, both O1TURN and PDIOR outperform single-path routing by 33%, with PDIOR performing at 96% of dynamic-VCA O1TURN (Figure 2). More interestingly, PDIOR can significantly outperform dynamic-VCA O1TURN (bit-complement, shuffle, H.264 profile); this is because PDIOR mitigates head-of-line blocking and improves load balancing by independently adjusting the switching frequency for each flow. The relative performance of PDIOR benefits even more from ameliorating head-of-line blocking when there are more virtual channels at each port: while PDIOR does slightly better for all benchmarks, the performance of dynamic DOR and O1TURN degrades because of increased head-of-line blocking (Figure 3).

5.3 Latency analysis

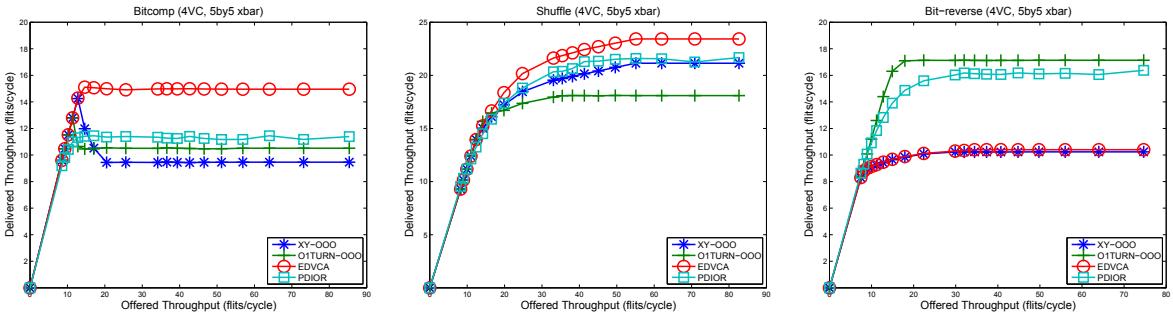
While we focus more on throughput performance, we reasoned that PDIOR may have longer packet latency than dynamic-VCA DOR/O1TURN as PDIOR prevents packets from being injected until the previous switching packet has been acknowledged by the destination node.

Figure 4 shows the average end-to-end packet latency with 4 virtual channels per port. The PDIOR latency plot is clearly different from the others: in some cases its latency shoots up at a lower throughput than O1TURN (bit-complement) and in other cases it offers reasonable latency up to a much higher throughput compared to O1TURN (shuffle, transpose, H.264); in almost all cases, however, the latency under PDIOR increases more gradually.

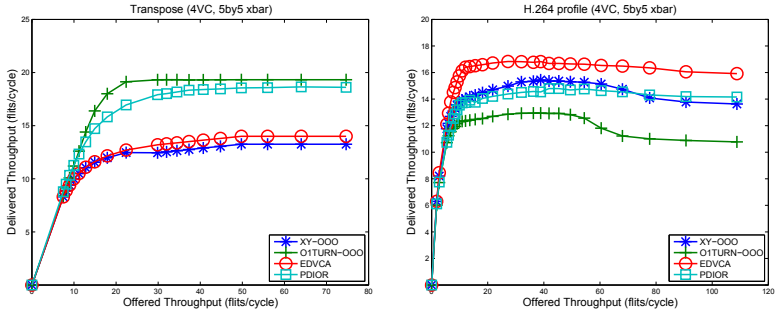
This is a side-effect of the PDIOR duty cycle of *on_time* and *off_time*: the offered rate during *on_time* is higher than the delivered rate on average even if the network is not fully saturated, because some packets can be delivered in *off_time*. Packets offered in *on_time* can thus experience increased end-to-end latency, and PDIOR latency starts increasing before O1TURN even if its saturated throughput is similar (bit-reverse and transpose), or even higher (shuffle). On the other hand, latency under PDIOR increases only gradually, as offered packets are eventually delivered during

Topology	8×8 2D mesh
Routing	DOR–XY(OoO), O1TURN(OoO), EDVCA(XY), PDIOR(O1TURN)
Link bandwith	1 flit/cycle
Crossbar size	5-by-5
VCs per port	4, 8
VC buffer size	8 flits
Avg. packet size	8 flits
Traffic workload	transpose, shuffle, bit-complement, bit-reverse, H.264 decoder profile
Warmup cycles	240,000
Analyzed cycles	960,000
Burstiness model	Markov modulated process

1: Network configuration summary

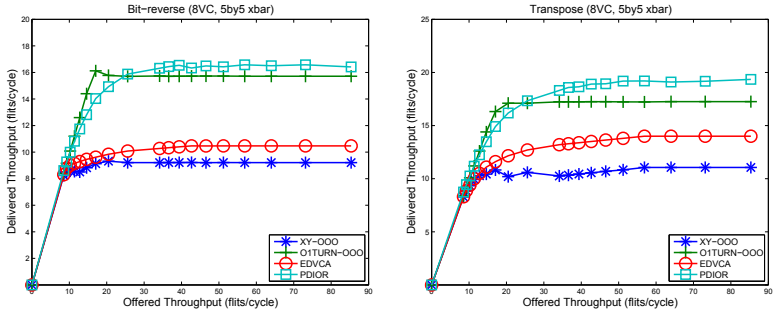


(a) bit-complement (b) shuffle (c) bit-reverse



(d) transpose (e) H.264 profile

2: Throughput of PDIOR (inorder) vs. DOR/O1TURN (out-of-order) and EDVCA (inorder) under 4 VCs.



(a) bit-reverse (b) transpose

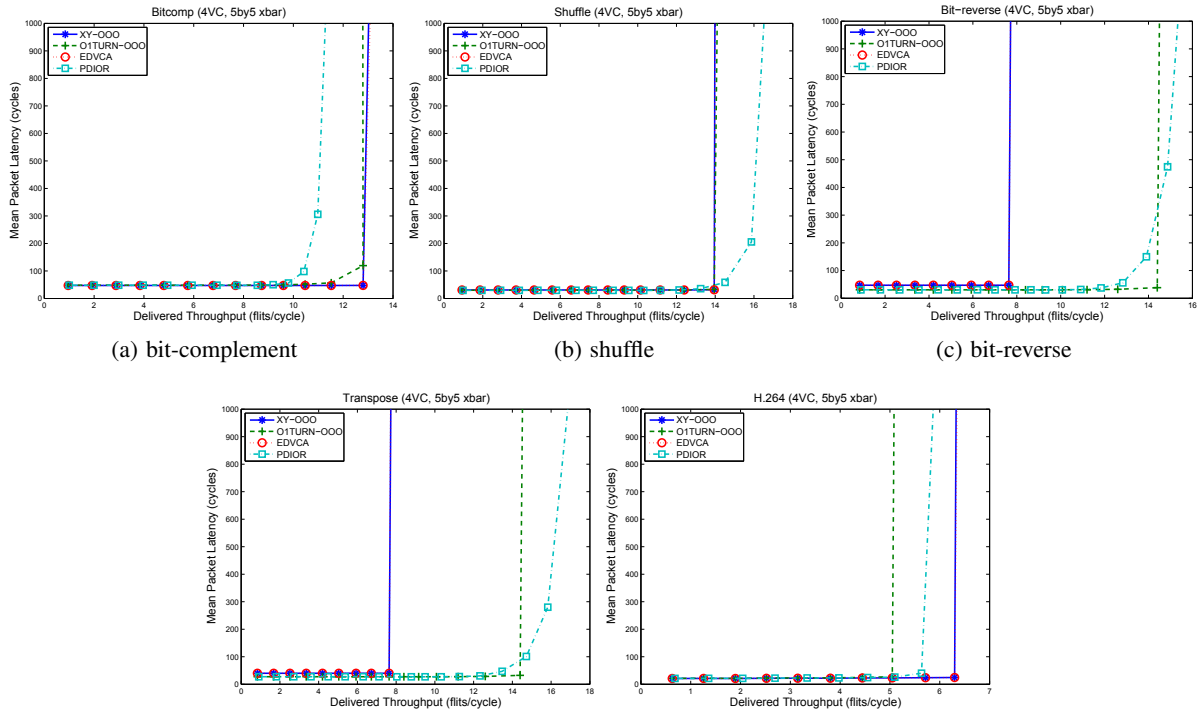
3: Throughput of PDIOR (inorder) vs. DOR/O1TURN (out-of-order) and EDVCA (inorder) under 8 VCs.

off_time (bit-complement is an exception here because the throughput of dynamic-VCA DOR and O1TURN is unstable: at lower offered rate the delivered rate of DOR and O1TURN is higher than PDIOR so the latency seems to be lower than PDIOR, but at higher offered rates, DOR/O1TURN throughput goes down so the latency at saturation jumps to infinity at a lower delivered rate than PDIOR). At higher offered rates, on the other hand, the actual injection rates of PDIOR become lower than O1TURN because injection pauses during *off_time*, and the latency of PDIOR at saturation becomes lower than O1TURN even when the saturated throughput is worse than O1TURN (e.g., bit-reverse and transpose).

Note that, for consistency with the throughput results, the plots for dynamic-VCA DOR and O1TURN *exclude* latencies associated with packet reordering. In both of these out-of-order schemes, implementing inorder delivery would contribute significantly to the observed latency, as it would add both the latency of waiting for out-of-order packets and the latency of reordering the packets.

6. CONCLUSION

Although applications that require packets to arrive in the order in which they were sent are ubiquitous, guaranteeing inorder packet delivery has received comparatively little attention in routing algorithm design, and, with the exception of single-VC dimension-order routing and static VC assignment, has generally been relegated to a higher level of abstraction. As ultra-fast on-chip networks become common, however, buffer-based packet reordering can become a significant bottleneck. Moreover, existing inorder network schemes do not apply to multi-path routing which can enhance throughput performance of the network. We have proposed Path Diverse In Order Routing, which extends the inorder guarantee of EDVCA to routing algorithms where each flow may be routed via multiple paths. Ensuring inorder delivery under various routing algorithms at the network level obviates the need for expensive buffers and retransmission logic, promising better performance at a lower cost than a traditional higher-level store-and-reorder scheme in the niche of fast on-chip networks.



(a) bit-complement (b) shuffle (c) bit-reverse

(d) transpose (e) H.264 profile

4: Average end-to-end latencies of PDIOR, dynamic-VCA DOR/O1TURN and EDVCA under 4 VCs. Reorder latency is not included for dynamic-VCA/O1TURN.

7. REFERENCES

[1] Arnab Banerjee and Simon Moore. Flow-Aware Allocation for On-Chip Networks. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip*, May 2009.

[2] Ge-Ming Chiu. The Odd-Even Turn Model for Adaptive Routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[3] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[4] Roman Gindin, Israel Cidon, and Idit Keidar. NoC-based FPGA: Architecture and routing. In *Proc. of the ACM/IEEE Int. Symp. on Networks-on-Chip (NOCS)*, May 2007.

[5] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, 1994.

[6] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, September 1996.

[7] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[8] Michel Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten van Dijk, and Srinivas Devadas. Application-Aware Deadlock-Free Oblivious Routing. In *Proceedings of the Int'l Symposium on Computer Architecture*, June 2009.

[9] M. Lis, K. S. Shim, M. H. Cho, and S. Devadas. Guaranteed in-order packet delivery using Exclusive Dynamic Virtual Channel Allocation. Technical Report CSAIL-TR-2009-036 (http://hdl.handle.net/1721.1/46353), Massachusetts Institute of Technology, August 2009.

[10] S. Murali, D. Atienza, L. Benini, and G. De Micheli. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of DAC 2006*, pages 845–848, July 2006.

[11] Ted Nesson and S. Lennart Johnsson. ROMM Routing: A Class of Efficient Minimal Routing Algorithms. In *in Proc. Parallel Computer Routing and Communication Workshop*, pages 185–199, 1994.

[12] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'95*, pages 275–287, 1995.

[13] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA 2005)*, pages 432–443, 2005.

[14] K. S. Shim, M. H. Cho, M. Kinsy, T. Wen, M. Lis, G. E. Suh, and S. Devadas. Static Virtual Channel Allocation in Oblivious Routing. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip*, May 2009.

[15] William Thies, Michał Karczmarek, and Saman P. Amarasinghe. StreamIt: A Language for Streaming Applications. In *Proceedings of the International Conference on Compiler Construction*, LCNS, pages 179–196, Grenoble, France, 2002.

[16] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, 1981.