A

Improving Energy Efficiency of Hierarchical Rings via Deflection Routing

Rachata Ausavarungnirun
rachata@cmu.edu
Chris Fallin‡
cfallin@cmu.edu
Xiangyao Yu†
yxy@mit.edu
Kevin Chang
kevincha@cmu.edu
Greg Nazario
gnazario@cmu.edu
Reetuparna Das§
reetudas@umich.edu
Gabriel H. Loh∗
gabriel.loh@amd.com
Onur Mutlu
onur@cmu.edu

Computer Architecture Lab (CALCM)
Carnegie Mellon University

†Massachusetts Institute of Technology
§University of Michigan
‡Intel Corporation
∗Advance Micro Devices

SAFARI Technical Report No. 2014-002

Ring topologies are popular for current on-chip interconnection networks. Rings are simple to implement as they require no in-ring buffering or flow control, and they are effective for small-scale multi-core architectures. However, the rapid trend toward higher core counts quickly makes traditional ring topologies impractical: average distance in the network increases linearly with node count, bisection bandwidth does not increase. Hierarchical ring networks, which hierarchically connect multiple levels of rings, have been proposed in the past to improve the scalability of ring interconnects, but past hierarchical ring designs sacrifice some of the key benefits of rings by reintroducing more complex in-ring buffering and buffered flow control. Our goal in this paper is to design a new hierarchical ring interconnect that can maintain most of the simplicity of traditional ring designs (i.e., no in-ring buffering or buffered flow control) while achieving high scalability as more complex buffered hierarchical ring designs.

To this end, we revisit the concept of a hierarchical-ring network-on-chip. Our design, called **HiRD** (Hierarchical Rings with Deflection), includes critical features that enable us to mostly maintain the simplicity of traditional simple ring topologies while providing higher energy efficiency and scalability. *First*, HiRD does not have *any* buffering or buffered flow control within individual rings, and requires only a small amount of buffering between the ring hierarchy levels. When inter-ring buffers are full, our design simply *deflects* flits so that they circle the ring and try again, which eliminates the need for in-ring buffering. *Second*, we introduce two simple mechanisms that together provide an end-to-end delivery guarantee within the entire network (despite any deflections that occur) without impacting the critical path or latency of the vast majority of network traffic.

Our experimental evaluations on a wide variety of multiprogrammed and multithreaded workloads and synthetic traffic patterns show that HiRD attains equal or better performance at better energy efficiency than multiple versions of both a previous hierarchical ring design and a traditional single ring design. We also extensively analyze our design's characteristics and injection and delivery guarantees. We conclude that HiRD can be a compelling design point that allows higher energy efficiency and scalability while retaining the simplicity and appeal of conventional ring-based designs.

Categories and Subject Descriptors: C.1.2 [**Multiple Data Stream Architectures**]: Interconnection Architectures

General Terms: Design, Performance

Additional Key Words and Phrases: Network-on-chip, Hierarchical Ring Design, Energy Efficient Interconnect Designs

1. INTRODUCTION

Interconnect scalability, performance, and energy efficiency are first-order concerns in the design of future CMPs (chip multiprocessors). As CMPs are built with greater numbers of cores, centralized interconnects (such as crossbars or shared buses) are no longer scalable. The Network-on-Chip (NoC) is the most commonly-proposed solution [Dally and Towles 2001]: cores exchange packets over a network consisting of network switches and links arranged in some topology.

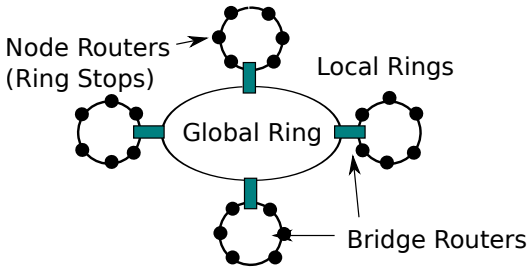


Fig. 1: A traditional hierarchical ring design [Ravindran and Stumm 1997; Zhang and Yan 1995; Harmacher and Jiang 2001; Ravindran and Stumm 1998; Grindley et al. 2000] allows "local rings" with simple node routers to scale by connecting to a "global ring" via bridge routers.

Mainstream commercial CMPs today most commonly use *ring*-based interconnects. Rings are a well-known network topology [Dally and Towles 2004], and the idea behind a ring topology is very simple: all routers (also called "ring stops") are connected by a loop that carries network traffic. At each router, new traffic can be injected into the ring, and traffic in the ring can be removed from the ring when it reaches its destination. When traffic is traveling on the ring, it continues uninterrupted until it reaches its destination. A ring router thus *needs no in-ring buffering or flow control* because it prioritizes on-ring traffic. In addition, the router's datapath is very simple compared to a mesh router, because the router has fewer inputs and requires no large, power-inefficient crossbars; typically it consists only of several MUXes to allow traffic to enter and leave, and one pipeline register. Its latency is typically only one cycle, because no routing decisions or output port allocations are necessary (other than removing traffic from the ring when it arrives). Because of these advantages, several prototype and commercial multicore processors have utilized ring interconnects: the Intel Larrabee [Seiler et al. 2008], IBM Cell [Pham et al. 2006], and more recently, the Intel Sandy Bridge [Intel Corporation 2011].

Unfortunately, rings suffer from a fundamental scaling problem because a ring's bisection bandwidth does not scale with the number of nodes in the network. Building more rings, or a wider ring, serves as a stopgap measure but increases the cost of every router on the ring in proportion to the bandwidth increase. As commercial CMPs continue to increase core counts, a new network design will be needed that balances the simplicity and low overhead of rings with the scalability of more complex topologies.

A hybrid design is possible: rings can be constructed in a *hierarchy* such that groups of nodes share a simple ring interconnect, and these "local" rings are joined by one or more "global" rings. Figure 1 shows an example of such a *hierarchical ring* design. Past works [Ravindran and Stumm 1997; Zhang and Yan 1995; Harmacher and Jiang 2001; Ravindran and Stumm 1998; Grindley et al. 2000] proposed hierarchical rings as a scalable network. These proposals join rings with *bridge routers*, which reside on multiple rings and transfer traffic between rings. This design was shown to yield good performance and scalability [Ravindran and Stumm 1997]. The state-of-the-art design [Ravindran and Stumm 1997] requires *flow control and buffering* at every node router (ring stop), because a ring transfer can make one ring back up and stall when another ring is congested. While this previously proposed hierarchical ring is superior to a conventional mesh, the reintroduction of in-ring buffering and flow control nullifies one of the primary advantages of using ring networks in the first place (i.e., the lack of buffering and buffered flow control within each ring).

Our goal in this work is to design a ring-based topology that is simpler and more efficient than prior ring-based topologies. To this end, our design uses simple ring networks that do not introduce any in-ring buffering or flow control. Like past proposals, we utilize a hierarchy-of-rings topology to achieve higher scalability. However, beyond the topological similarities, our design is very different in how traffic is handled within individual rings and between different levels of rings. We introduce a new *bridge router* microarchitecture that facilitates the transfer of packets from one ring to another. It is in these, and *only* these, limited number of bridge routers where we require any buffering.

Our key idea is to allow a bridge router with a full buffer to *deflect* packets. Rather than requiring buffering and flow control in the ring, packets simply cycle through the network and try again. While deflection-based, bufferless networks have been proposed and evaluated in the past [Baran 1964; Hillis 1989; Smith 1981; Alverson et al. 1990; Moscibroda and Mutlu 2009; Gómez et al. 2008a], our approach is effectively an elegant hybridization of bufferless (rings) and buffered (bridge routers) styles. To prevent packets from potentially deflecting around a ring arbitrarily many times (i.e., to prevent livelock), we introduce two new mechanisms, the *injection guarantee* and the *transfer guarantee*, that ensure packet delivery even for adversarial/pathological conditions (as we discuss in §4 and evaluate with worst-case traffic in §6.3).

This simple hierarchical ring design, which we call *HiRD* (for Hierarchical Rings with Deflection), provides a more scalable network architecture while retaining the key simplicities of ring networks (no buffering or flow control within each ring). We show in our evaluations that HiRD provides better performance, lower power, and better energy efficiency with respect to the state-of-the-art buffered hierarchical ring design [Ravindran and Stumm 1997].

In summary, **our major contributions** are:

— We propose a new, low-cost, hierarchical ring NoC design based on very simple router microarchitectures that achieve single-cycle latencies. This design, *HiRD*, places an ordinary ring router (without flow control or buffering) at every network node, connects local rings with global rings using *bridge routers*, which have minimal buffering and use deflection rather than buffered flow control for inter-ring transfers.
— We provide new mechanisms for *guaranteed delivery of traffic* ensuring that inter-ring transfers do not cause livelock or deadlock, even in the worst case.
— We qualitatively and quantitatively compare HiRD to several state-of-the-art NoC designs. We show competitive performance to these baselines, with better energy efficiency than all prior designs, including, most importantly, the state-of-the-art hierarchical ring design with in-ring buffering and buffered flow control [Ravindran and Stumm 1997]. We conclude that HiRD represents a compelling design point for future many-core interconnects by achieving higher performance while maintaining most of the simplicity of traditional ring-based designs.

2. MOTIVATION

In this section, we first show how ring routers can be very simple, but not scalable. We then describe past work in hierarchical rings and show how a previous design obtained performance scalability with rings, but required reintroducing in-ring flow control and buffering, thereby obviating the primary benefits of rings. These two observations – rings are simple but do not scale, and a hierarchy of rings scales but increases complexity – motivate our *simple* hierarchical ring interconnect, HiRD.

2.1. Simplicity in Ring-Based Interconnects

Ring interconnects are attractive in current small-to-medium-scale commercial CMPs because ring routers (ring stops) are simple, which leads to smaller die area and energy overhead. In its simplest form, a ring router needs to perform only two functions: injecting new traffic into the ring, and removing traffic from the ring when it has arrived at its destination. Figure 2 a) depicts the router datapath and control logic (at a high level) necessary to implement this functionality. For every incoming *flit* (unit of data transfer as wide as one link), the router only needs to determine whether the flit stays on the ring or exits at this node. Then, if a flit is waiting to inject, the router checks whether a flit is already present on the ring in this timeslot, and if not, injects the new flit using the in-ring MUX.

Rings achieve very good energy efficiency at low-to-medium core counts [Kim and Kim 2009]; However, the simplicity advantage of a ring-based network, rings have a fundamental *scalability* limit: a ring stops scaling at fewer nodes because its bisection bandwidth is *constant* (proportional only to link width) and the average hop count (which translates to latency for a packet) increases linearly with the number of nodes. (In the worst case for a bidirectional ring, a packet visits half the network on its way to its destination, and a quarter on average.)

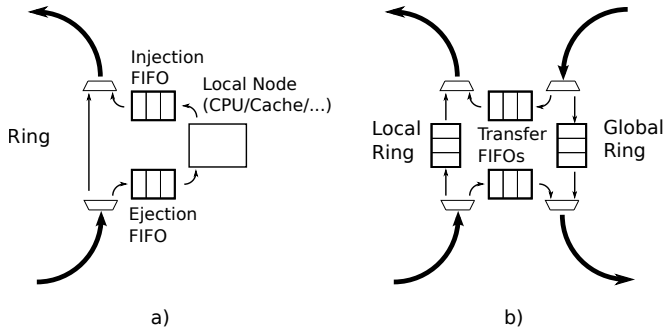


Fig. 2: a) A ring router. b) Buffered hierarchical ring detail, as proposed in prior work [Ravindran and Stumm 1997]: one *bridge router* which connects two rings.

2.2. Hierarchical Rings for Scalability

Fortunately, past work has observed that *hierarchy* allows for additional scalability in many interconnects: in ring-based designs [Ravindran and Stumm 1997; Zhang and Yan 1995; Harmacher and Jiang 2001; Ravindran and Stumm 1998; Grindley et al. 2000], with hierarchical buses [Udipi et al. 2010], and with hierarchical meshes [Das et al. 2009]. The state-of-the-art hierarchical ring design [Ravindran and Stumm 1997] in particular reports promising results by combining local rings with one or more levels of higher-level rings, which we refer to as global rings, that connect lower level rings together. Rings of different levels are joined by Inter-Ring Interfaces (IRIs), which we call "bridge routers" in this work. Figure 2 b) graphically depicts the details of one bridge router in the previously proposed buffered hierarchical ring network.

Unfortunately, connecting multiple rings via bridge routers introduces a new problem. Recall that injecting new traffic into a ring requires an open slot in that ring. If the ring is completely full with flits (i.e., a router attempting to inject a new flit must instead pass through a flit already on the ring every cycle), then no new traffic will be injected. But, a bridge router must inject transferring flits into those flits' destination ring in exactly the same manner as if they were newly entering the network. If the ring on one end of the bridge router is completely full (cannot accept any new flits), and the transfer FIFO into that ring is also full, then any other flit requiring a transfer must *block* its current ring. In other words, ring transfers create new dependences between adjacent rings, which creates the need for end-to-end flow control. This flow control forces every node router (ring stop) to have an in-ring FIFO and flow control logic, which increases energy and die area overhead and significantly reduces the appeal of a simple ring-based design. Table I compares the power consumption for a previously proposed hierarchical ring design (Buffered HRing) and a bufferless hierarchical ring design on a system with 16 nodes using DSENT 0.91 [Sun et al. 2012] and a 45nm technology commercial design library. In the examined idealized bufferless design, each ring has no in-ring buffers, but there are buffers between the hierarchy levels. When a packet needs a buffer that is full, it gets deflected and circles its local ring to try again. Clearly, eliminating in-ring buffers in a hierarchical ring network can reduce power and area significantly.

Metric	Buffered HRing	Bufferless HRing
Normalized power	1	0.535
Normalized area	1	0.495

Table I: Power and area comparison (16-node network).

However, simply removing in-ring buffers can introduce livelock, as a deflected packet may circle its local ring indefinitely. Our goal in this work is to introduce a hierarchical ring design that maintains simplicity and low cost, while ensuring livelock and deadlock freedom for packets.

3. HIRD: SIMPLE HIERARCHICAL RINGS WITH DEFLECTION

In this section, we describe the operation of our network design *HiRD*, or Hierarchical Rings with Deflection. HiRD is built on several basic operation principles:

(1) Every node (e.g., CPU, cache slice, or memory controller) resides on one *local ring*, and connects to one *node router* on that ring.
(2) Node routers operate exactly like routers (ring stops) in a single-ring interconnect: locally-destined flits are removed from the ring, other flits are passed through, and new flits can inject whenever there is a free slot (no flit present in a given cycle). There is no buffering or flow control within any local ring; flits are buffered only in ring pipeline registers. Node routers have a single-cycle latency.
(3) Local rings are connected to one or more levels of *global rings* to form a tree hierarchy.
(4) Rings are joined via *bridge routers*. A bridge router has a node-router-like interface on each of the two rings which it connects, and has a set of transfer FIFOs (one in each direction) between the rings.
(5) Bridge routers consume flits that require a transfer whenever the respective transfer FIFO has available space. The head flit in a transfer FIFO can inject into its new ring whenever there is a free slot (exactly as with new flit injections). When a flit requires a transfer but the respective transfer FIFO is full, the flit remains in its current ring. It will circle the ring and try again next time it encounters the correct bridge router (this is a *deflection*).

By using *deflections* rather than buffering and blocking flow control to manage ring transfers, HiRD retains node router simplicity, unlike past hierarchical ring network designs. This change comes at the cost of potential livelock (if flits are forced to deflect forever). We introduce two mechanisms to provide a deterministic guarantee of livelock-free operation in §4.

While deflection-based bufferless routing has been previously proposed and evaluated for a variety of off-chip and on-chip interconnection networks (e.g., [Baran 1964; Moscibroda and Mutlu 2009; Gómez et al. 2008a; Fallin et al. 2011]), deflections are trivially implementable in a ring: if deflection occurs, the flit continues circulating in the ring. Contrast this to past deflection-based schemes that operated on mesh networks where multiple incoming flits may need to be deflected among a multitude of possible out-bound ports, leading to much more circuit complexity in the router microarchitecture, as shown by [Fallin et al. 2011; Hayenga et al. 2009; Michelogiannakis et al. 2010]). Our application of deflection to rings leads to a simple and elegant embodiment of bufferless routing.

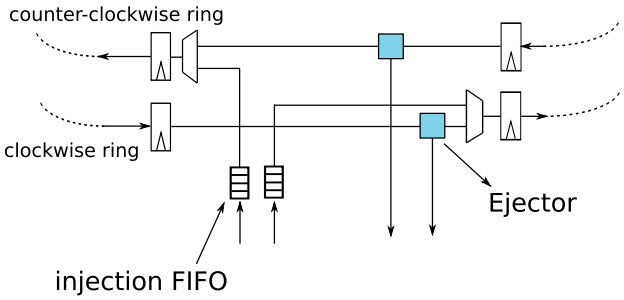


Fig. 3: Node router.

3.1. Node Router Operation

At each node on a local ring, we place a single node router, shown in Figure 3. A node router is very simple: it passes through circulating traffic, allows new traffic to enter the ring through a MUX, and allows traffic to leave the ring when it arrives at its destination. Each router contains one pipeline register for the router stage, and one pipeline register for link traversal, so the router latency is

exactly one cycle and the per-hop latency is two cycles. Such a design as this is very common in ring-based and ring-like designs (e.g., [Kim 2009]).

As flits enter the router on the ring, they first travel to the ejector. Because we use bidirectional rings, each node router has two ejectors, one per direction[1]. Note that the flits constituting a packet may arrive out-of-order and at widely separated times. Re-assembly into packets is thus necessary. Packets are re-assembled and reassembly buffers are managed using the Retransmit-Once scheme, borrowed from the CHIPPER bufferless router design [Fallin et al. 2011]. With this scheme, receivers reassemble packets in-place in MSHRs (Miss-Status Handling Registers [Kroft 1981]), eliminating the need for separate reassembly buffers. The key idea in Retransmit-Once is to avoid ejection backpressure-induced deadlocks by ensuring that all arriving flits are consumed immediately at their receiver nodes. When a flit from a new packet arrives, it allocates a new reassembly buffer slot if available. If no slot is available, the receiver drops the flit and sets a bit in a retransmit queue which corresponds to the sender and transaction ID of the dropped flit. Eventually, when a buffer slot becomes available at the receiver, the receiver reserves the slot for a sender/transaction ID in its retransmit queue and requests a retransmit from the sender. Thus, all traffic arriving at a node is consumed (or dropped) immediately, so ejection never places backpressure on the ring. Retransmit-Once hence avoids protocol-level deadlock [Fallin et al. 2011]. Furthermore, it ensures that a ring full of flits always drains, thus ensuring forward progress (as we will describe more fully in §4).

After locally-destined traffic is removed from the ring, the remaining traffic travels to the injection stage. At this stage, the router looks for "empty slots," or cycles where no flit is present on the ring, and injects new flits into the ring whenever they are queued for injection. The injector is even simpler than the ejector, because it only needs to find cycles where no flit is present and insert new flits in these slots. Note that we implement two separate injection buffers (FIFOs), one per ring direction; thus, two flits can be injected into the network in a single cycle. A flit enqueues for injection in the direction that yields a shorter traversal toward its destination.

3.2. Bridge Routers

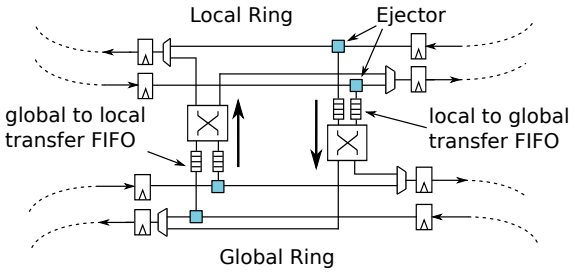
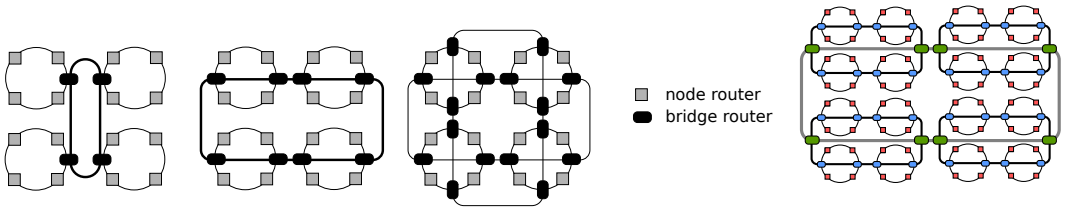


Fig. 4: Bridge router.

Second, the *bridge routers* connect a local ring and a global ring, or a global ring with a higher-level global ring (if there are more than two levels of hierarchy). A high-level block diagram of a bridge router is shown in Figure 4. A bridge router resembles two node routers, one on each of two rings, connected by FIFO buffers in both directions. When a flit arrives on one ring that requires a transfer to the other ring (according to the routing function described below in §3.3), it can leave its current ring and wait in a FIFO as long as there is space available. These *transfer FIFOs* exist so that a transferring flit's arrival need not be perfectly aligned with a free slot on the

[1]For simplicity, we assume that up to two ejected flits can be accepted by the processor or reassembly buffers in a single cycle. For a fair comparison, we also implement two-flit-per-cycle ejection in our mesh-based baselines.



(a) 4-, 8-, and 16-bridge hierarchical ring designs.

(b) Three-level hierarchy (8x8).

Fig. 5: Hierarchical ring design of HiRD

destination ring. However, this transfer FIFO will sometimes fill. In that case, if any flit arrives that requires a transfer, the bridge router simply does not remove the flit from its current ring; the flit will continue to travel around the ring, and will eventually come back to the bridge router, at which point there may be an open slot available in the transfer FIFO. This is analogous to a *deflection* in hot-potato routing [Baran 1964], also known as deflection routing, and has been used in recent on-chip mesh interconnect designs to resolve contention [Moscibroda and Mutlu 2009; Fallin et al. 2011; Fallin et al. 2012; Tota et al. 2006]. Note that to ensure that flits are *eventually* delivered, despite any deflections that may occur, we introduce two *guarantee mechanisms* in §4. Finally, note that deflections may cause flits to arrive out-of-order (this is fundamental to any non-minimal adaptively-routed network). Because we use Retransmit-Once [Fallin et al. 2011], packet reassembly works despite out-of-order arrival.

The bridge router uses *crossbars* to allow a flit ejecting from either ring direction in a bidirectional ring to enqueue for injection in either direction in the adjoining ring. When a flit transfers, it picks the ring direction that gives a shorter distance, as in a node router. However, these crossbars actually allow for a more general case: the bridge router can actually join several rings together by using larger crossbars. For our network topology, we use hierarchical rings. We use wider global rings than local rings (analogous to a *fat tree* [Hillis 1989]) for performance reasons. These wider rings perform logically as separate rings as wide as one flit. Although not shown in the figure for simplicity, the bridge router in such a case uses a larger crossbar and has one ring interface (including transfer FIFO) per ring-lane in the wide global ring. The bridge router then load-balances flits between rings when multiple lanes are available. (The crossbar and transfer FIFOs are fully modeled in our evaluations.)

When building a two-level design, there are many different arrangements of global rings and bridge routers that can efficiently link the local rings together. The size (bandwidth) and number of global rings, as well as the number of bridges between each local and global ring, must be tuned like any other system design trade-off. In this work, we study three particular configurations, shown in Figure 5a. We refer to these designs by the number of bridge routers in total: 4-bridge, 8-bridge, and 16-bridge. The first two designs contain a single global ring, with either one or two bridge routers for each local ring, respectively. The last, 16-bridge, design contains *two* global rings with two bridge routers per local-global ring combination. As we will show in §6.6, this design choice has a significant positive impact on system performance. It will also negatively impact network power and total router area. Outside of that sensitivity study, we assume an 8-bridge design for the remainder of this paper.

Also, note that the hierarchical structure that we propose can be extended to more than two levels; in general, rings can be combined into a hierarchy with bridge routers. We use a 3-level hierarchy, illustrated in Figure 5b, to build a 64-node network.

Finally, in order to address a potential deadlock case (which will be explained in detail in §4), bridge routers implement a special *Swap Rule*. The Swap Rule states that when the flit that just arrived on each ring requires a transfer to the other ring, the flits can be *swapped*, bypassing the transfer FIFOs altogether. This requires a bypass datapath (which is fully modeled in our hardware evaluations) but ensures correct operation in the case when transfer FIFOs in both directions are full. Only one swap needs to occur in any given cycle, even when the bridge router connects to a

wide global ring. Note that because the swap rule requires this bypass path, the behavior is always active (it would be more difficult to definitively identify a deadlock and enable the behavior only in that special case). The Swap Rule may cause flits to arrive out-of-order when some are bypassed in this way, but the network already delivers flits out-of-order, so correctness is not compromised.

3.3. Routing

Finally, we briefly address routing. Because a hierarchical ring design is fundamentally a *tree*, routing is very simple: when a flit is destined for a node in another part of the hierarchy, it first travels *up* the tree (to more global levels) until it reaches a common ancestor of its source and its destination, and then it travels *down* the tree to its destination. Concretely, each node's address can be written as a series of parts, or digits, corresponding to each level of the hierarchy (these trivially could be bitfields in a node ID). A ring can be identified by the common prefix of all routers on that ring; the root global ring has a null (empty) prefix, and local rings have prefixes consisting of all digits but the last one. If a flit's destination does not match the prefix of the ring it is on, it takes any bridge router to a more global ring. If a flit's destination does match the prefix of the ring it is on (meaning that it is traveling down to a more local levels), it takes any bridge router which connects to a next level of more local ring, until it finally reaches the local ring of its destination and ejects at the node with a full address match.

4. GUARANTEED DELIVERY: CORRECTNESS IN HIERARCHICAL RING INTERCONNECTS

In order for the system to operate correctly, the interconnect must guarantee that every flit is eventually delivered to its destination. HiRD ensures correct operation through two mechanisms that provide two guarantees: the *injection guarantee* and the *transfer guarantee*. The injection guarantee ensures that any flit waiting to inject into a ring will eventually be able to enter that ring. The transfer guarantee ensures that any flit waiting to enter a bridge router's transfer queue will eventually be granted a slot in that queue. These guarantees must be provided explicitly because HiRD's deflection-based ring transfers might otherwise cause a flit to remain in the network for an unbounded time. Together, the mechanisms that we introduce to ensure these two properties provide an end-to-end guarantee that all flits will be delivered.

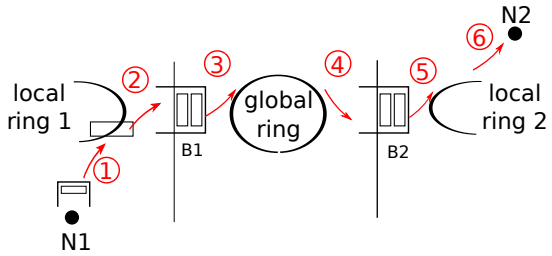


Fig. 6: The need for the injection and transfer guarantees: contention experienced by a flit during its journey.

To understand the need for each guarantee, let us consider a simple example, shown in Figure 6. A flit is enqueued for network injection at node N1 on the leftmost local ring. This flit is destined for node N2 on the rightmost local ring; hence, it must traverse the leftmost local ring, then the global ring in the center of the figure, followed by the rightmost local ring. The flit transfers rings twice, at the two bridge routers B1 and B2 shown in the figure. The figure also indicates the six points (labeled as circled points 1 – 6) at which the flit moves from a queue to a ring or vice-versa: the flit first enters N1's injection queue, transfers to the leftmost local ring (Point 1), the bridge router B1 (Point 2), the global ring (Point 3), the bridge router B2 (Point 4), the rightmost local ring (Point 5), and finally the destination node N2.

In the worst case, when the network is heavily contended, the flit could wait for an unbounded amount of time at the five Points 1 – 5. First, recall that to enter any ring, a flit must wait for an

empty slot on that ring (because the traffic on the ring continues along the ring once it has entered, and thus has higher priority than any new traffic). Because of this, the flit traveling from node N1 to N2 could wait for an arbitrarily long time at Points 1, 3, and 5 if no other mechanism intercedes. This first problem is one of *injection starvation*, and we address it with the *injection guarantee* mechanism described below. Second, recall that a flit which needs to transfer from one ring to another via a bridge router enters that bridge router's queue, but if the bridge router's queue is full, then the transferring flit must make another trip around its current ring and try again when it next encounters a bridge router. Because of this rule, the flit traveling from N1 to N2 could be *deflected* an arbitrarily large number of times at Points 2 and 4 (at entry to bridge routers B1 and B2) if no other mechanism intercedes. This second problem is one of *transfer starvation*, and we address it with the *transfer guarantee* mechanism described below.

Our goal in this section is to demonstrate that HiRD provides both the injection guarantee (§4.1) and transfer guarantee (§4.2) mechanisms. We qualitatively argue correctness in §4.3, and quantitatively evaluate both mechanisms in §6.3.

4.1. Preventing Injection Starvation: Injection Guarantee

The *injection guarantee* ensures that every router on a ring can eventually inject a flit. This guarantee is provided by a very simple throttling-based mechanism: when any node is starved (cannot inject a flit) past a threshold number of cycles, it asserts a signal to a global controller, which then throttles injection from every other node. No new traffic will enter the network while this throttling state is active. All existing flits in the network will eventually drain, and the starved node will be able to finally inject its new flit. At that time, the starved node deasserts its throttling request signal to the global controller, and the global controller subsequently allows all other nodes to resume normal operation. While we will more formally argue correctness below, this injection guarantee mechanism is correct simply because the throttling remains active until the starved node's flit is injected; in the most extreme case, all flits in the network will drain, and the starved node will be able to inject (in most cases, the problem resolves before all flits drain). Thus, HiRD provides an injection guarantee.

Note that this injection guarantee can be implemented in a hierarchical manner to improve scalability. In the hierarchical implementation, each individual local ring in the network monitors only its own injection and throttles injection locally if any node in it is starved. After a threshold number of cycles[2], if any node in the ring still cannot inject, the bridge routers connected to that ring starts sending throttling signals to any other ring in the next level of the ring hierarchy it is connected to. In the worst case, every local ring stops accepting flits and all the flits in the network drain and eliminate any potential livelock or deadlock. Designing the delivery guarantee this way will only require an additional 1-bit wire in each ring and small design overhead at the bridge router to propagate the throttling signal across hierarchies. In our evaluation, we faithfully model this hierarchical design.

4.2. Ensuring Ring Transfers: Transfer Guarantee

The *transfer guarantee* ensures that any flit waiting to transfer from its current ring to another ring via a bridge router will eventually be able to enter that bridge router's queue. Such a guarantee is non-trivial because the bridge router's queue is finite, and when the destination ring is congested, a slot may become available in the queue only infrequently. In the worst case, a flit in one ring may circulate indefinitely, finding a bridge router to its destination ring with a completely full queue each time it arrives at the bridge router. The transfer guarantee ensures that any such circulating flit will eventually be granted an open slot in the bridge router's transfer queue. Note in particular that this guarantee is *separate from* the injection guarantee: while the injection guarantee ensures that the bridge router will be able to inject flits from its transfer queue into the destination ring (and hence, have open slots in its transfer queue eventually), these open transfer slots may not be distributed *fairly* to flits circulating on a ring waiting to transfer through the bridge router. In other words, some

[2]In our evaluation we set this threshold to be 100 cycles.

flit may always be "unlucky" and never enter the bridge router if slots open at the wrong time. The transfer guarantee addresses this problem.

In order to ensure that any flit waiting to transfer out of a ring eventually enters its required bridge router, each bridge router *observes a particular slot on its source ring* and monitors for flits that are "stuck" for more than a threshold number of retries. (To observe one "slot," the bridge router simply examines the flit in its ring pipeline register once every N cycles, where N is the latency for a flit to travel around the ring once.) If any flit circulates in its ring more than this threshold number of times, the bridge router reserves the next open available entry in its transfer queue for this flit (in other words, it will refuse to accept other flits for transfer until the "stuck" flit enters the queue). Because of the injection guarantee, the head of the transfer queue must inject into the destination ring eventually, hence an entry must become available eventually, and the stuck flit will then take the entry in the transfer queue the next time it arrives at the bridge router. Finally, the slot which the bridge router observes rotates around its source ring: whenever the bridge router observes a slot the second time, if the flit that occupied the slot on first observation is no longer present (i.e., successfully transferred out of the ring or ejected at its destination), then the bridge router begins to observe the *next* slot (the slot that arrives in the next cycle). In this way, every slot in the ring is observed eventually, and any stuck flit will thus eventually be granted a transfer.

4.3. Putting it Together: Guaranteed Delivery

We now formally prove the end-to-end delivery guarantee for any flit in the network given the injection and transfer guarantees described above.

Before we prove the correctness of these mechanisms in detail, it is helpful to summarize the basic operation of the network once more. A flit can inject into a ring whenever a free slot is present in the ring at the injecting router (except when the injecting router is throttled by the injection guarantee mechanism). A flit can eject at its destination whenever it arrives, and destinations always consume flits as soon as they arrive (which is ensured despite finite reassembly buffers using the Retransmit-Once [Fallin et al. 2011], as already described in §3.1). A flit transfers between rings via a transfer queue in a bridge router, first leaving its source ring to wait in the queue and then injecting into its destination ring when at the head of the queue, and can enter a transfer queue whenever there is a free entry in that transfer queue (except when the entry is reserved for another flit by the transfer guarantee mechanism). Finally, when two flits at opposite ends of a bridge router each desire to to transfer through the bridge router, the *swap rule* allows these flits to exchange places directly, bypassing the queues (and ensuring forward progress).

Our proof is structured as follows: we first argue that if no new flits enter the network, then the network will drain in finite time. The injection guarantee ensures that any flit can enter the network. Then, using the injection guarantee, transfer guarantee, the swap rule, and the fact that the network is hierarchical, any flit in the network can eventually reach any ring in the network (and hence, its final destination ring). Because all flits in a ring continue to circulate that ring, and all nodes on a ring must consume any flits that are destined for that node, final delivery is ensured once a flit reaches its final destination ring.

Network drains in finite time: Assume no new flits enter the network (for now). A flit could only be stuck in the network indefinitely if transferring flits create a cyclic dependence between completely full rings. Otherwise, if there are no dependence cycles, then if one ring is full and cannot accept new flits because other rings will not accept *its* flits, then eventually there must be some ring which depends on no other ring (e.g., a local ring with all locally-destined flits), and this ring will drain first, followed by the others feeding into it. However, because the network is hierarchical (i.e., a tree), the only cyclic dependences possible are between rings that are immediate parent and child (e.g., global ring and local ring, in a two-level hierarchy). The *Swap Rule* ensures that when a parent and child ring are each full of flits that require transfer to the other ring, then transfer is always possible, and forward progress will be ensured. Note in particular that we do not require the injection or transfer guarantee for the network to drain. Only the swap rule is necessary to ensure that no livelock will occur.

Any node can inject: Now that we have shown that the network will drain if no new flits are injected, it is easy to see that the injection guarantee ensures that any node can eventually inject a flit: if any node is starved, then all nodes are throttled, no new flit enters the network, and the network must eventually drain (as we just showed), at which point the starved node will encounter a completely empty network into which to inject its flit. (It likely will be able to inject before the network is completely empty, but in the worst case, the guarantee is ensured in this way.)

All flits can transfer rings and reach their destination rings: With the injection guarantee in place, the transfer guarantee can be shown to provide its stated guarantee as follows: because of the injection guarantee, a transfer queue in a bridge router will always inject its head flit in finite time, hence will have an open entry to accept a new transferring flit in finite time. All that is necessary to ensure that *all* transferring flits eventually succeed in their transfers is that *any* flit stuck for long enough gets an available entry in the transfer queue. But the transfer guarantee does exactly this by observing ring slots in sequence and reserving a transfer queue entries when a flit becomes stuck in a ring. Because the mechanism will eventually observe every slot in the ring, all flits will be allowed to make their transfers eventually. Hence all flits can continue to transfer rings until reaching their destination rings (and thus, their final destinations).

4.4. Hardware Cost

The mechanisms proposed here have a low hardware overhead. To implement the injection guarantee, one counter is required for each injection point. This counter tracks how many cycles have elapsed while injection is starved, and is reset whenever a flit is successfully injected. Routers communicate with the throttling arbitration logic with only two wires, one to signal blocked injection and one control line that throttles the router. The wiring is done hierarchically instead of globally to minimize the wiring cost. Because the correctness of the algorithm does not depend on the delay of these wires, and the injection guarantee mechanism is activated only rarely (in fact, never for our realistic workloads), the signaling and central coordinator need not be optimized for speed. To provide the transfer guarantee, each bridge router implements "observer" functionality for each of the two rings it sits on, and the observer consists only of three small counters (to track the current timeslot being observed, the current timeslot at the ring pipeline register in this router, and the number of times the observed flit has circled the ring) and a small amount of control logic. Importantly, note that neither mechanism impacts the router critical path nor affects the router datapath (which dominates energy and area).

5. EVALUATION METHODOLOGY

We perform our evaluations using a cycle-accurate simulator of a CMP system with interconnect to provide application-level performance results. Details are given in Tables II and III. In brief, the simulator is an in-house x86 CMP simulator. Each CPU core faithfully models stalls due to memory latency, and a cache hierarchy with a distributed cache coherence protocol (based on SGI Origin [Laudon and Lenoski 1997]) is modeled on top of the cycle-accurate interconnect model. Each core is out-of-order and has a set of MSHRs (Miss-Status Handling Registers [Kroft 1981], which track outstanding cache-miss requests) realistically, including the additional request-buffering overhead, possible stalls due to a congested network, and the overhead from retransmit once [Fallin et al. 2011]. The shared last-level L2 cache is *perfect* (always hits), which removes the effects of memory latency and increases network load (penalizing our deflection-based design relative to other, higher-capacity designs). This methodology ensures a rigorous and isolated evaluation of NoC capacity for especially cache resident workloads, and has also been used in other studies [Moscibroda and Mutlu 2009; Fallin et al. 2011]. Instruction traces for the simulator are taken using a Pintool [Luk et al. 2005] on representative portions of SPEC CPU2006 workloads [Henning 2006].

We mainly compare to a single bidirectional ring and a state-of-the-art buffered hierarchical ring [Ravindran and Stumm 1997]. For completeness, we also provide a comparison against a typical buffered mesh topology [Dally 1992; Dally and Seitz 1987], a bufferless mesh (CHIPPER [Fallin et al. 2011]), and a flattened butterfly topology [Kim and Dally 2007] in Section 6.8. To fairly compare networks, we hold the bisection bandwidth constant. This comparison is fair because the mod-

Parameter	Setting
System topology	CPU core and shared cache slice at every node
Core model	Out-of-order, 128-entry ROB, 16 MSHRs (simultaneous outstanding requests)
Private L1 cache	64 KB, 4-way associative, 32-byte block size
Shared L2 cache	Perfect (always hits) to stress the network and penalize our reduced-capacity deflection-based design; cache-block-interleaved mapping[3]
Cache coherence	Directory-based protocol (based on SGI Origin [Laudon and Lenoski 1997; Culler et al. 1999]), directory entries co-located with shared cache blocks
Simulation length	5M instructions warm-up, 25M instructions active execution per node[4]

Table II: Simulation and system configuration parameters.

Parameter	Network	Setting
Interconnect Links	Single Ring	Bidirectional, **4x4:** 64-bit and 128-bit width, **8x8:** 128-bit and 256-bit width
	Buffered HRing	Bidirectional, **4x4:** 3-cycle per-hop latency; 64-bit local and 128-bit global rings, **8x8:** three-level hierarchy, 4x4 parameters, with second-level rings connected by 256-bit third-level rings
	HiRD	**4x4:** 2-cycle (local), 3-cycle (global) per-hop latency; 64-bit local ring, 128-bit global ring; **8x8:** 4x4 parameters, with second-level rings connected by 256-bit third-level ring.
Router	Single Ring	1-cycle per-hop latency (as in [Kim and Kim 2009])
	Buffered HRing	Node (NIC) and bridge (IRI) routers based on [Ravindran and Stumm 1997]; 4-flit in-ring and transfer FIFOs. Bidirectional links of dual-flit width (for fair comparison with our design). Bubble flow control [Carrión et al. 1997] for deadlock freedom.
	HiRD	local-to-global buffer depth 1, global-to-local buffer depth 4

Table III: Network parameters.

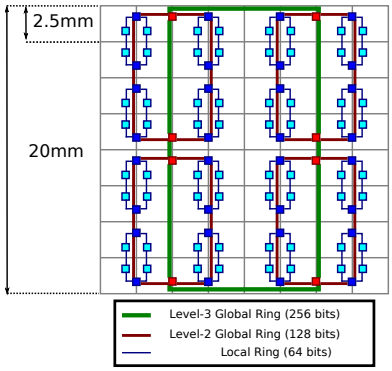


Fig. 7: Assumed floorplan for HiRD 3-level (64-node) network. Two-level (16-node) network consists of one quadrant of this floorplan.

eled hardware overhead (power, die area and timing) scales up with increased bisection bandwidth (so no design is unfairly boosted over another). Also, note that while there are many possible ways to optimize each baseline (such as congestion control, adaptive routing schemes, and careful parameter tuning), we assume a fairly typical aggressive configuration for each – e.g., our buffered mesh router has a two-cycle router latency, assuming speculative arbitration [Dally and Towles 2004], and uses buffer bypassing [Wang et al. 2003].

Data Mapping: We map data in a cache block interleaved way to different L2 shared cache. This mapping is agnostic to the underlying locality. As a result, it does not exploit the low latency data access in the local ring. One can design systematically better mapping in order to keep frequently used data in the local ring [Lee et al. 2011]. However, such mechanism is orthogonal to our proposal and can be applied in all ring-based network designs.

Application & Synthetic Workloads: We evaluate the system design with both multiprogrammed and multithreaded scenarios. In a multiprogrammed scenario, the system is run with a set of 60 multiprogrammed workloads. Each workload consists of one single-threaded instance of a SPEC CPU2006 benchmark on each core, for a total of either 16 (4x4) or 64 (8x8) benchmark instances per workload. Multiprogrammed workloads such as these are representative of many common workloads for large CMPs. Workloads are constructed at varying network intensities as follows: first, benchmarks are split into three classes (Low, Medium and High) by L1 cache miss intensity (which correlates directly with network injection rate), such that benchmarks with less than 5 misses per thousand instructions (MPKI) are "Low-intensity," between 5 and 50 are "Medium-intensity," and above 50 MPKI are "High-intensity." Workloads are then constructed by randomly selecting a certain number of benchmarks from each category. We form workload sets with four intensity mixes: High, Medium, Medium-Low, and Low, with 15 workloads in each (the average network injection rates for each category are 0.47, 0.32, 0.18, and 0.03 flits/node/cycle, respectively).

For multithreaded evaluations, we use GraphChi implementation of the GraphLab framework [Kyrola et al. 2012; Low et al. 2010]. The implementation we use is designed to run efficiently on multi-core systems. The workload consists of Twitter Community Detection (CD), Twitter Page Rank (PR), Twitter Connected Components (CC), Twitter Triangle Counting (TC) [Kwak et al. 2010], and Graph500 Breadth First Search (BFS). We simulated the representative portion of each workload and each workload has a working set size of greater than 151.3 MB. On every simulation of these multithreaded workloads, we warm up the cache with the first 5 million instructions, then we run the remaining code of the representative portion.

To show that our conclusions are robust across particular applications or coherence protocol traffic patterns, we also evaluate each network using synthetic uniform random traffic to demonstrate its fundamental capacity.

Energy & Area: We model router and link energy and area by individually modeling the crossbar, pipeline registers buffers, control logic and other datapath components. For links, buffers and datapath elements, we use DSENT 0.91 [Sun et al. 2012]. Control logic is modeled in Verilog. Both energy and area are calculated based on a 45nm technology.

We assume a 2.5 mm link length for single ring designs. For the hierarchical ring design, we assume 1 mm links between local-ring routers, because the four routers on a local ring can be placed at four corners that meet in a tiled design; global-ring links are assumed to be 5.0 mm, because they span across two tiles on average if local rings are placed in the center of each four-tile quadrant. Third-level global ring links are assumed to be 10mm in the 8x8 evaluations. This floorplan is illustrated in more detail in Figure 7.

Application Evaluation Metrics: For multiprogrammed workloads, we present application performance results using the commonly-used Weighted Speedup metric [Snavely and Tullsen 2000] (representing system throughput [Eyerman and Eeckhout 2008]), which calculates performance of each application in a multiprogrammed mix relative to how well the application performs when running alone on the system: $WS = \sum_{i=1}^{N} \frac{IPC_i^{shared}}{IPC_i^{alone}}$. For multithreaded workloads we measure performance using the speedup of the actual execution time, defined as the number of cycles each workload takes to finish running the representative portion of the work. This allows the fair comparison against different network design and ensuring that the amount of work is similar and ensuring that all thread level dependencies (e.g. synchronization) are taken into account.

6. EVALUATION

6.1. Ring-based Network Designs

Multiprogrammed workloads

Figure 8 shows performance (weighted speedup normalized per node), power (total network power normalized per node), and energy-efficiency (perf./power) for 16-node and 64-node HiRD

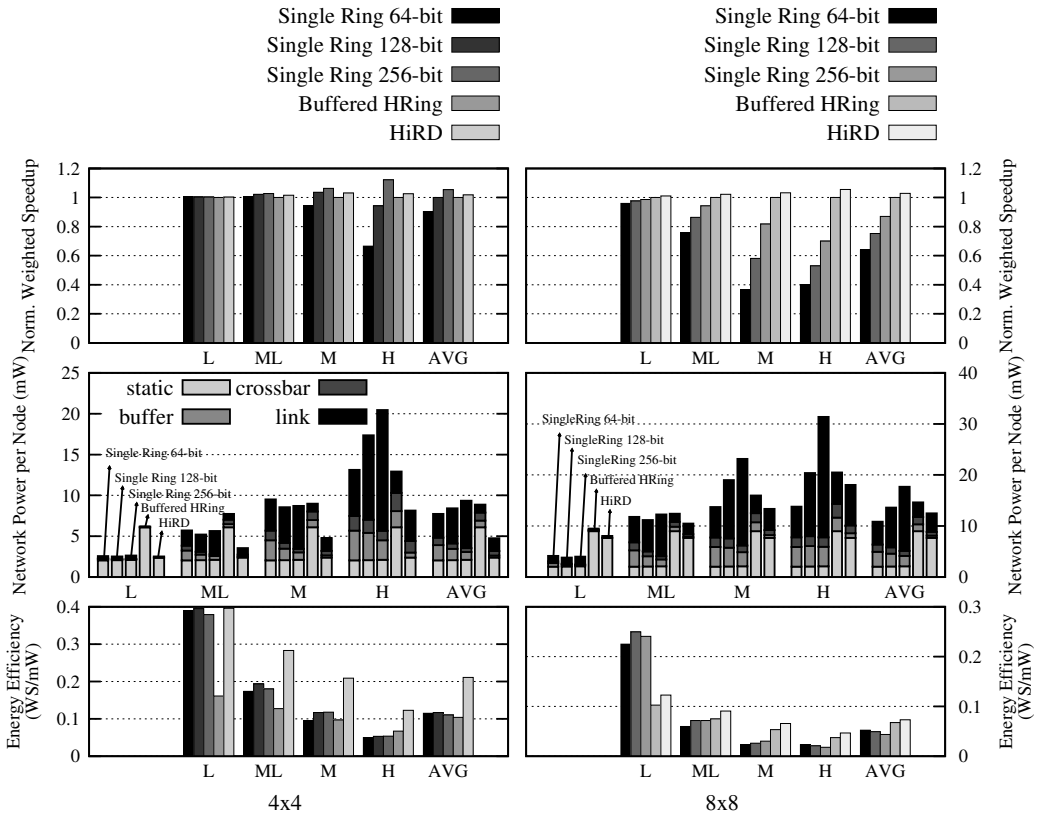


Fig. 8: HiRD as compared to buffered hierarchical rings and a single-ring network.

and buffered hierarchical rings, using identical topologies, as well as a single ring (with different bisection bandwidths).[5] Several major conclusions are in order:

1. A hierarchical topology yields significant performance advantages over a single ring (i) when network load is high and/or (ii) when the network scales to many nodes. As the data shown, the buffered hierarchical ring improves performance by 7% (and HiRD by 10%) in high-load workloads at 16 nodes compared to a single ring with 128-bit links. The hierarchical design also reduces power because hop count is reduced. Therefore, link power reduces significantly with respect to a single ring. On average in the 8x8 configuration, the buffered hierarchical ring network obtains 15.6% better application performance than the single ring with 256-bit links, while HiRD attains 18.2%.

2. Compared to the buffered hierarchical ring, HiRD has significantly lower network power. On average, HiRD reduces total network power (links and routers) by 46.5% (4x4) or 14.7% (8x8) relative to this baseline. This reduction in turn yields significantly better energy efficiency (lower energy consumption for buffers and slightly higher links energy). Overall, HiRD is the most energy-efficient of the ring-based designs evaluated in this paper in both 4x4 and 8x8 network sizes.

3. While scaling the link bandwidth increases the performance of a single ring network, the network power increases 25.9% when the link bandwidth increases from 64-bit to 128-bit and 15.7% when the link bandwidth increases from 128-bit to 256-bit because of dynamic energy coming off from wider links. In addition, scaling the link bandwidth is not a scalable solution as a single ring network performs worse than the bufferred hierarchical ring baseline even when a 256-bit link is used.

[5]Since our goal is to provide a better ring design, our main comparisons are to ring networks. Section 6.8 provides an evaluation of other topologies.

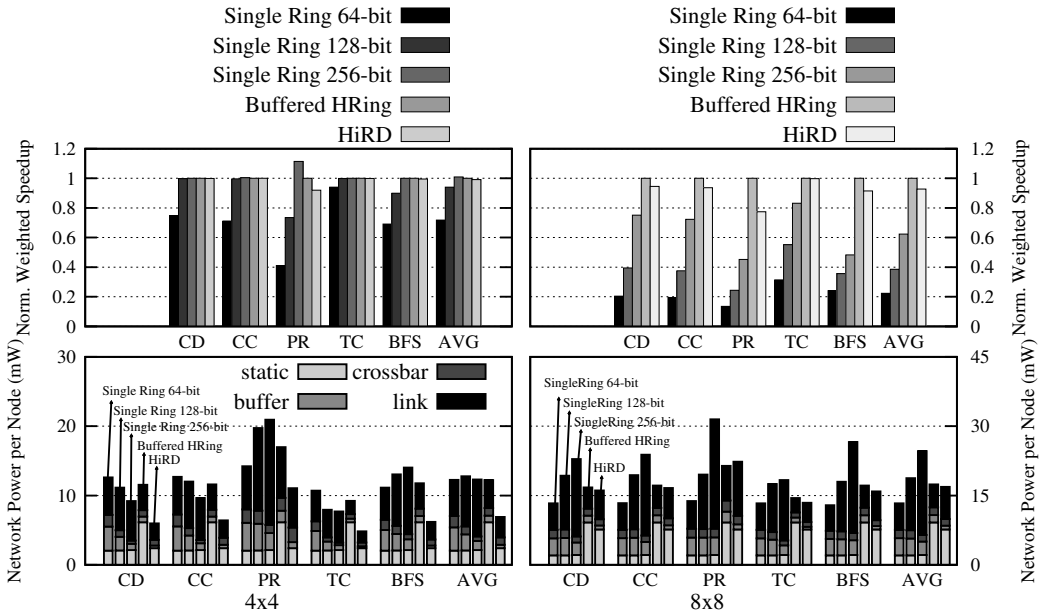


Fig. 9: HiRD as compared to buffered hierarchical rings and a single-ring network.

Multithreaded workloads

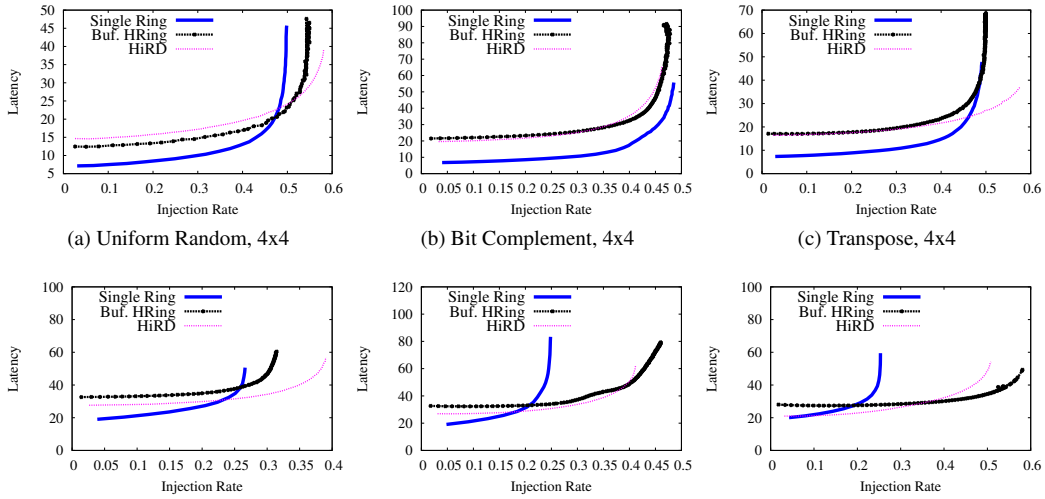
Figure 9 shows the performance and power of multithreaded applications of HiRD compared a buffered hierarchical ring and a single-ring network for both 16-node and 64-node systems. On average, HiRD performs 0.1% (4x4) and 0.73% (8x8) worse than the buffered hierarchical ring. However, on average, HiRD consumes 43.8% (4x4) and 3.1% (8x8) less power, leading to higher energy efficiency.

Both the buffered hierarchical ring and HiRD outperform single ring networks, both in performance and energy efficiency, and the gap increases as we scale the size of the network.

Even though HiRD performs competitively with a buffered hierarchical ring network in most cases, HiRD performs poorly on the Page Ranking application. We observe that Page Ranking generates more non-local network traffic. As a result, the advantage of HiRD where the local-ring latency is lower does not provide speedup for Page Ranking. In addition, Page Ranking also has higher network traffic, causing more congestion in the network (we observe 17.3% higher average network latency for HiRD in an 8x8 network), which causes additional performance drop for HiRD. However, it is potentially possible to use a different number of bridge routers as illustrated in Figure 5a, to improve the performance of HiRD. We will provide this analysis in Section 6.6. Additionally, it is possible to apply a locality-aware cache mapping technique [Lee et al. 2011] in order to take advantage of lower local-ring latency in HiRD.

6.2. Synthetic-Traffic Network Behavior

Figure 10 shows the average latency as a function of injection rate for buffered and bufferless mesh routers, a single-ring design, the buffered hierarchical ring, and HiRD in 16 and 64-node systems. We show uniform random, transpose and bit complement traffic patterns [Dally and Towles 2004]. Sweeps on injection rate terminate at network saturation. The buffered hierarchical ring saturates at a similar point to HiRD but maintains a slightly lower average latency because it avoids transfer deflections. In contrast to these high-capacity designs, the single ring saturates at a lower injection rate. As network size scales to 8x8, HiRD performs significantly better relative to single rings, because the hierarchy reduces the cross-chip latency while preserving bisection bandwidth. HiRD also performs better than Buffered HRing because of two reasons. First, HiRD is able to allow higher peak utilization (91%) than Buffered HRing (71%) on the global rings. We observed that



(a) Uniform Random, 4x4 (b) Bit Complement, 4x4 (c) Transpose, 4x4

(d) Uniform Random, 8x8 (e) Bit Complement, 8x8 (f) Transpose, 8x8

Fig. 10: Synthetic-traffic evaluations for 4x4 and 8x8 networks.

when flits have equal distance in a clock-wise and counter clock-wise direction, Buffered HRing has to send flits to one direction in order to avoid deadlock while deflections in HiRD allow flits to travel in both directions, leading to better overall network utilization. Second, at high injection rates, the transfer guarantee (Section 4) starts throttling the network, disallowing future flits to be injected into the network until the existing flits arrive at their destinations. This reduces congestion in the network and allows HiRD to saturate after a buffered hierarchical ring design.

6.3. Injection and Transfer Guarantees

In this subsection, we study HiRD's behavior under a worst-case synthetic traffic pattern that triggers the injection and transfer guarantees and demonstrates that they are necessary for correct operation, and that they work as designed.

Traffic Pattern: In the worst-case traffic pattern, all nodes on three rings in a two-level (16-node) hierarchy inject traffic (we call these rings Ring A, Ring B, and Ring C). Rings A, B, and C have bridge routers adjacent to each other, in that order, on the single global ring. All nodes in Ring A continuously inject flits that are addressed to nodes in Ring C, and all nodes in Ring C likewise inject flits to nodes in Ring A. This creates heavy traffic on the global ring across the point at which Ring B's bridge router connects. All nodes on Ring B continuously inject flits (whenever they are able) addressed to another ring elsewhere in the network. However, because Rings A and C continuously inject flits, Ring B's bridge router will not be able to transfer any flits to the global ring in the steady state (unless another mechanism such as the throttling mechanism in Section 4 intercedes).

Configuration	Transfer FIFO Wait (cycles)	Deflections/Retries	Network Throughput (flits/node/cycle)		
	(avg/max)	(avg/max)	Ring A	Ring B	Ring C
Without Guarantees	2.5 / 299670	6.0 / 49983	0.164	0.000	0.163
With Guarantees	1.2 / 66	2.8 / 18	0.133	0.084	0.121

Table IV: Results of worst-case traffic pattern without and with injection/transfer guarantees enabled.

Results: Table IV shows three pertinent metrics on the network running the described traffic pattern: average network throughput (flits/node/cycle) for nodes on Rings A, B, and C, the maximum time

Configuration	Transfer FIFO Wait (cycles) (avg/max)	Deflections/Retries (avg/max)
Without guarantees	3.3 / 169	3.7 / 19
With guarantees	0.76 / 72	0.7 / 8

Table V: Effect from transfer guarantee mechanism on real workloads.

(in cycles) spent by any one flit at the head of a transfer FIFO, and the maximum number of times any flit is deflected and has to circle a ring to try again. These metrics are reported with the injection and transfer guarantee mechanisms disabled and enabled. The experiment is run with the synthetic traffic pattern for 300K cycles.

The results show that without the injection and transfer guarantees, Ring B is completely starved and cannot transfer any flits onto the global ring. This is confirmed by the maximum transfer FIFO wait time, which is almost the entire length of the simulation. In other words, once steady state is reached, no flit ever transfers out of Ring B. Once the transfer FIFO in Ring B's bridge router fills, the local ring fills with more flits awaiting a transfer, and these flits are continuously deflected. Hence the maximum deflection count is very high. Without the injection or transfer guarantee, the network does *not* ensure forward progress for these flits. In contrast, when the injection and transfer guarantees are enabled, (i) Ring B's bridge router is able to inject flits into the global ring and (ii) Ring B's bridge router fairly picks flits from its local ring to place into its transfer FIFO. The maximum transfer FIFO wait time and maximum deflection count are now bounded, and nodes on all rings receive network injection throughput. Thus, the guarantees are both necessary and sufficient to ensure deterministic forward progress for all flits in the network.

Real Applications: Table V shows the effect of the transfer guarantee mechanism on real applications in a 4x4 network. Average transfer FIFO wait shows the average number of cycles that a flit waits on average in the transfer FIFO across all 60 workloads. Maximum transfer FIFO wait shows the maximum observed latency across all workloads. As illustrated in Table V, a few flits can experience very high wait time when there is no transfer guarantee, and our transfer guarantee mechanism reduces both average and maximum FIFO waits[6]. In addition, we observe that our transfer guarantee mechanism not only provides livelock- and deadlock-freedom but also provides lower maximum wait time in the transfer FIFO for each flit because the guarantee provides a form of throttling when the network is congested. This similar observation is also discussed in many previous network-on-chip works that use source throttling to improve the performance of the network [Thottethodi et al. 2001; Baydal et al. 2005; Nychis et al. 2010; Chang et al. 2012].

6.4. Network Latency and Latency Distribution

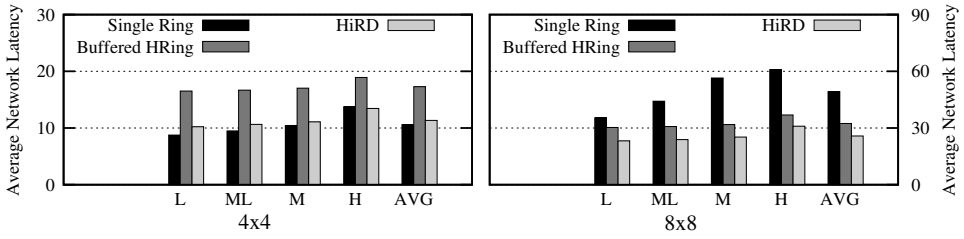


Fig. 11: Average network latency for 4x4 and 8x8 networks.

Figure 11 shows average network latency. This plot shows that our proposal can reduce the network latency by having a faster local-ring hop latency compared to other ring-based designs. Additionally, we found that, for all real workloads, the number of deflections we observed is always

[6]As the network scales to 64 nodes, we observe that the average wait in the transfer FIFO does not affect the overall performance significantly (additional 1.5 cycles per flit).

less than 3% of the total number of flits. Therefore, the benefit of our deflection based router design outweighs the extra cost of deflections compared to other ring-based router designs.

Fig. 12: Maximum network latency for 4x4 and 8x8 networks.

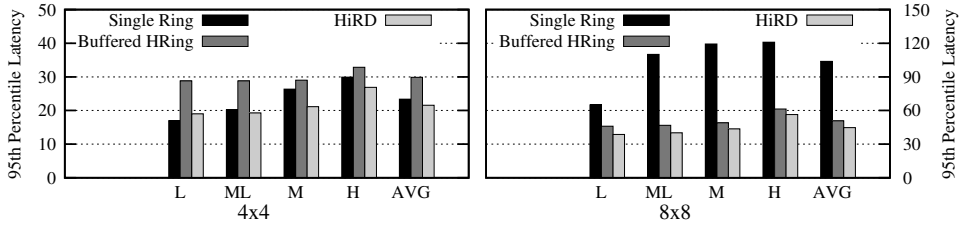


Fig. 13: 95th percentile latency for 4x4 and 8x8 networks.

In addition, Figure 12 shows the maximum latency and Figure 13 shows the 95th percentile latency for each network design. These two figures provide a quantitative evidence that the network is deadlock-free and livelock-free. Several additional observations are in order:

1. HiRD provides lower latency at the 95th percentile and the lowest latency observed in the network. This lower latency comes from our transfer guarantee mechanism that is applied when flits spend more than 100 cycles in each local ring, draining all flits in the network to their destination. This also means that HiRD improves the worst case latency that a flit can experience because none of the flits are severely slowed down.

2. While both HiRD and the buffered hierarchical ring have higher 95th percentile and maximum flit latency compared to a 64-bit single ring network, both hierarchical designs have 60.1% (buffered hierarchical ring) and 53.9% (HiRD) lower average network latency in an 8x8 network because hierarchy provides better scalability on average.

3. Maximum latency in the single ring is low because contention happens only at injection and ejection, as opposed to hierarchical designs where contention can also happen when flits travel through different hierarchies.

4. The transfer guarantee in HiRD also helps to significantly reduce the maximum latency observed by some flits compared to a buffered design because the guarantee enables the throttling of the network, thereby alleviating congestion. Reduced congestion leads to reduced maximum latency. This observation is confirmed with our synthetic traffic results shown in Section 6.2

6.5. Router Area and Timing

We show both critical path length and normalized die area for single-ring, buffered hierarchical ring, and HiRD, in Table VI. Area results are normalized to the buffered hierarchical ring baseline, and are reported for all routers required by a 16-node network (e.g., for HiRD, 16 node routers and 8 bridge routers).

Two observations are in order. First, HiRD reduces area relative to the buffered hierarchical ring routers, because the node router required at each network node is much simpler and does not require complex flow control logic. HiRD reduces total router area by 50.3% from the buffered hierarchical

Metric	Single-Ring	Buffered HRing	HiRD
Critical path length (ns)	0.33	0.87	0.61
Normalized area	0.281	1	0.497

Table VI: Total router area (16-node network) and critical path.

ring. Its area is higher than a single ring router because it contains buffers in bridge routers. However, the energy efficiency of HiRD and its performance at high load make up for this shortcoming. Second, the buffered hierarchical ring router's critical path is 42.6% longer than HiRD because its control logic must also handle flow control (it must check whether credits are available for a downstream buffer). The single-ring network has a higher operating frequency than HiRD because it does not need to accommodate ring transfers (but recall that this simplicity comes at the cost of poor performance at high load for the single ring).

6.6. Sensitivity Studies

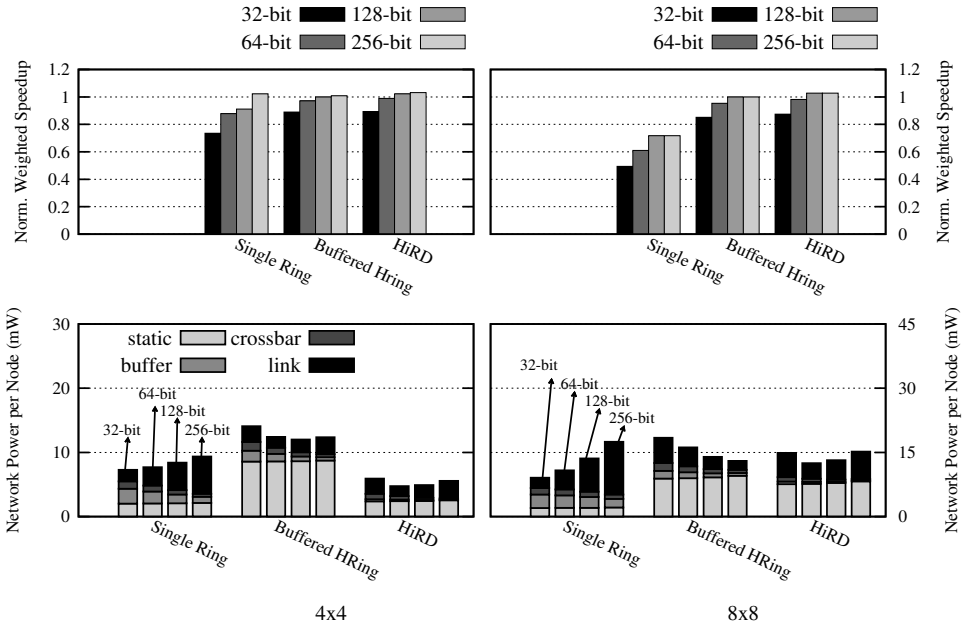


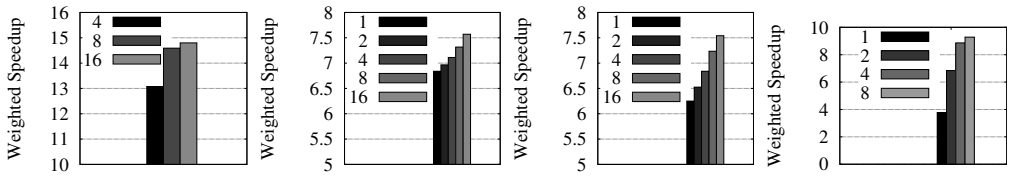
Fig. 14: Sensitivity to different link bandwidth for 4x4 and 8x8 networks.

6.6.1. Sensitivity to Link Bandwidth. The bandwidth of each link also has an effect on the performance of different network designs. In this subsection, we evaluate the effect of different link bandwidths on several ring-based networks by using 32- 64- and 128-bit links on all network designs. Figure 14 shows the performance and power consumption of each network design. As links get wider, the performance of each design increases. According to the result, HiRD performs slightly better than a buffered hierarchical ring design for almost all link bandwidths while maintaining much lower power consumption on a 4x4 network, and slightly lower power consumption on an 8x8 network.

Additionally, we observe that increasing link bandwidth can decrease the network power in a hierarchical design because lower link bandwidth causes more congestion in the network and leads to more dynamic buffer, crossbar and link power consumption due to additional deflections at the buffers. As the link bandwidth increases, congestion reduces, lowering dynamic power. However,

we observe that past a certain link bandwidth (e.g., 128 bits for buffered hierarchical ring and HiRD), congestion no longer reduces, because deflections at the buffers become the bottleneck instead. This leads to diminishing returns in performance yet increased dynamic power.

6.6.2. Sensitivity to Configuration Parameters. **Bridge Router Organization:** The number of bridge routers that connect the global ring(s) to the local rings has an important effect on system performance because the connection between local and global rings can limit bisection bandwidth. In Figure 5a, we showed three alternative arrangements for a 16-node network, with 4, 8, and 16 bridge routers. So far, we have assumed an 8-bridge design in 4x4-node systems, and a system with 8 bridge routers at each level in 8x8-node networks (Figure 5b). In Figure 15a, we show average performance across all workloads for a 4x4-node system with 4, 8, and 16 bridge routers. Buffer capacity is held constant. As shown, significant performance is lost if only 4 bridge routers are used (10.4% on average). However, doubling from 8 to 16 bridge routers gains only 1.4% performance on average. Thus, the 8-bridge design provides the best tradeoff of performance and network cost (power and area) overall in our evaluations.



(a) Number of bridge routers (b) Local-to-global bridge buffer (c) Global-to-local bridge buffer (d) Global-Local ring B/W ratio

Fig. 15: Performance sensitivity to buffer sizes and the global ring bandwidth in a 4x4 network.

Bridge Router Buffer Size: The size of the FIFO queues that are used to transfer flits between local and global rings can have an impact on performance if they are too small (and hence are often full and deflect transferring flits) or too large (and hence increase bridge router power and die area). We show the effect of local-to-global and global-to-local FIFO sizes in Figs. 15b and 15c, respectively, for the 8-bridge 4x4-node design. In both cases, increased buffer size leads to increased performance. However, performance is more sensitive to global-to-local buffer size (20.7% gain from 1 to 16-flit buffer size) than local-to-global size (10.7% performance gain from 1 to 16 flits), because in the 8-bridge configuration, the whole-loop latency around the global ring is slightly higher than in the local rings, so a global-to-local transfer retry is more expensive. For our evaluations, we use a 4-flit global-to-local and 1-flit local-to-global buffer per bridge router, which results in transfer deflection rates of 28.2% (global-to-local) and 34% (local-to-global).

Global Ring Bandwidth: Previous work on hierarchical-ring designs did not examine the impact of global-ring bandwidth on performance, but instead, assumed equal bandwidth in local and global rings [Ravindran and Stumm 1998]. In Figure 15d, we examine the sensitivity of system performance to global ring bandwidth relative to local ring bandwidth, for the all-High category of workloads (in order to stress bisection bandwidth). Each point in the plot is described by this bandwidth ratio. The local ring design is held constant while the width of the global ring is adjusted. If a ratio of 1:1 is assumed (leftmost bar), performance is significantly worse than the best possible design. Our main evaluations in 4x4 networks use a ratio of 2:1 (global:local) in order to provide equivalent bisection bandwidth to a 4x4 mesh baseline. Performance increases by 81.3% from a 1:1 ratio to the 2:1 ratio that we use. After a certain point, the global ring is no longer the bottleneck, and further global-ring bandwidth increases have little effect.

Delivery guarantee parameters: In Section 4, we introduced injection guarantee and ejection guarantee mechanisms to ensure every flit is eventually delivered. The injection guarantee mechanism takes a threshold parameter that specifies how long an injection can be blocked before action is

taken. Setting this parameter too low can have an adverse impact on performance, because the system throttles too aggressively and thus underutilizes the network. Our main evaluations use a 100-cycle threshold. For High-intensity workloads, performance drops 21.3% when using a threshold of only 1 cycle. From 10 cycles upward, variation in performance is at most 0.6%: the mechanism is invoked rarely enough that the exact threshold does not matter, only that it is finite (for correctness). In fact, for a 100-cycle threshold, the injection guarantee mechanism is never triggered in our realistic workloads. The mechanism is necessary only for corner-case correctness. In addition, we evaluate the impact of communication latency between routers and the coordinator. We find less than 0.1% variation in performance for latencies from 1 to 30 cycles (when parameters are set so that the mechanism becomes active); thus, low-cost, slow wires may be used for this mechanism.

The ejection guarantee takes a single threshold parameter: the number of times a flit is allowed to circle around a ring before action is taken. We find less than 0.4% variation in performance with a threshold from 1 to 16. Thus, the mechanism provides correctness in corner cases but is unimportant for performance in the common case.

6.7. Comparison Against Other Ring Configurations

Figure 16 highlights the energy-efficiency comparison of different ring-based design configurations by showing weighted speedup (Y axis) against power (X axis) for all evaluated 4x4 networks. HiRD is shown with the three different bridge-router configurations (described in Section 3.2). Every ring design is evaluated at various link bandwidths (32-, 64-, 128- and 256-bit links). The top-left is the ideal corner (high performance, low power). As the results show, at the same link bandwidth, all three configurations of HiRD are more energy efficient than the evaluated buffered hierarchical ring baseline designs at this network size.

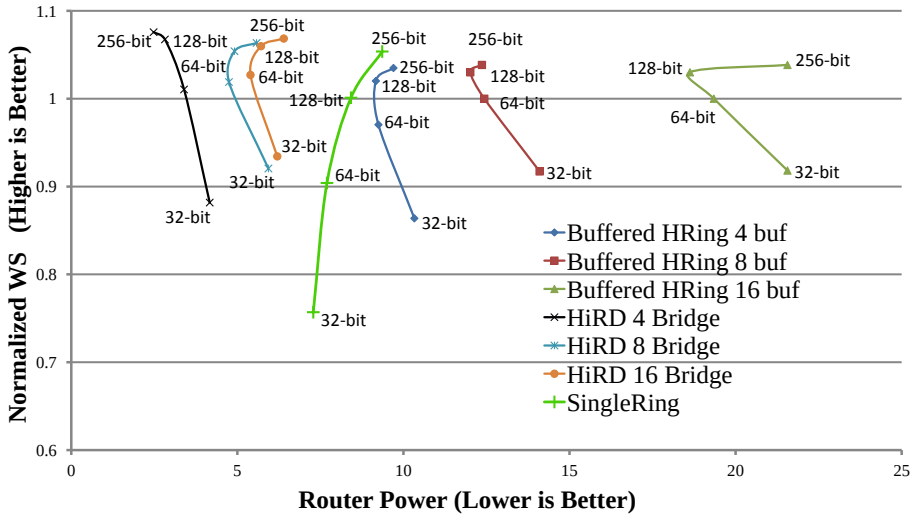


Fig. 16: Weighted speedup (Y) vs. power (X) for 4x4 networks.

We also observe that increasing link bandwidth can sometimes decrease router power as it reduces deflections in HiRD or lowers contention at the buffers in a buffered hierarchical ring design. However, once links are wide enough, this benefit diminishes for two reasons as discussed in Section 6.6 (links and crossbars consume more energy and packets arrive at the destination faster leading to higher power as more energy is consumed in less time).

Topologies	4x4		8x8	
	Norm. WS	Power (mWatts)	Norm. WS	Power (mWatts)
Single Ring	0.904	7.696	0.782	13.603
Buffered HRing	1	12.433	1	16.188
Buffered Mesh	1.025	11.947	1.091	13.454
CHIPPER	0.986	4.631	1.013	7.275
Flattened Butterfly	1.037	10.760	1.211	30.434
HiRD	1.020	4.746	1.066	12.480

Table VII: Evaluation for 4x4 and 8x8 networks against different network designs

6.8. Comparison Against Other Network Designs

For completeness, Table VII shows the comparison against several other network designs on 4x4 and 8x8 networks using the multiprogrammed workloads described in Section 5.

We compared our mechanism against a buffered mesh design with buffer bypassing. We configure the buffered mesh to have 4 VCs per port with 8 buffers per VC. We also compare our mechanism against CHIPPER [Fallin et al. 2011], a bufferless mesh network. We use 128-bit links for both designs. Additionally, we compare our mechanism against a flattened butterfly [Kim and Dally 2007] with 4 VCs per output port, 8 buffers per VC and 64-bit links. Our conclusions are as follows:

1. Compared to designs using the mesh topology, we observe that HiRD performs very closely to the buffered mesh design both for 4x4 and 8x8 network sizes while a buffered hierarchical ring design performs slightly worse compared to HiRD and buffered mesh designs. Additionally, HiRD performs better than CHIPPER in both 4x4 and 8x8 networks but CHIPPER consumes less power in an 8x8 design as there is no buffer in CHIPPER.

2. Compared to a flattened butterfly design, we observe that HiRD performs competitively with a flattened butterfly in a 4x4 network, but consumes lower router power. In an 8x8 network, HiRD does not scale as well as a flattened butterfly network and performs 11% worse than a flattened butterfly network but consumes 59% less power than the flattened butterfly design.

7. RELATED WORK

To our knowledge, HiRD is the first hierarchical ring design that uses simple, deflection-based ring transfers to eliminate the need for buffering while guaranty end-to-end packet delivery.

Hierarchical Interconnects: Hierarchical ring-based interconnect was proposed in a previous line of work [Ravindran and Stumm 1997; Zhang and Yan 1995; Harmacher and Jiang 2001; Ravindran and Stumm 1998; Grindley et al. 2000]. We have already extensively compared to past hierarchical ring proposals qualitatively and quantitatively. The major difference between our proposal and this previous work is that we propose deflection-based bridge routers with minimal buffering, and node routers with no buffering; in contrast, all of these previous works use routers with in-ring buffering, and use wormhole switching and flow control. This is analogous to the difference between buffered mesh routers [Dally 1992; Dally and Seitz 1987] and bufferless deflection mesh routers [Baran 1964; Tota et al. 2006; Moscibroda and Mutlu 2009; Fallin et al. 2011]. Our design leads to a significantly simpler router with reduced area and network power, as we show in this paper.

Udipi et al. proposed a hierarchical topology using global and local buses [Udipi et al. 2010]. While this work recognizes the benefits of hierarchy, our design builds upon a ring-based design instead of a bus-based design because a ring-based design provides better scalability. Das et al. examined several hierarchical designs, including a concentrated mesh (one mesh router shared by several nearby nodes) [Das et al. 2009].

A previous system, SCI (Scalable Coherent Interface) [Gustavson 1992], also uses rings, and can be configured in many topologies (including hierarchical rings). However, to handle buffer-full conditions, SCI NACKs and subsequently retransmits packets, whereas HiRD deflects only single flits (within one ring) drops, and does not require the sender to retransmit its flits. SCI was designed for off-chip interconnect, where tradeoffs in power and performance are very different than in on-chip interconnects. The KSR (Kendall Square Research) machine [Dunigan 1994] uses

a hierarchical ring design that resembles HiRD, yet these techniques are not disclosed in detail and, to our knowledge, have not been publicly evaluated in terms of energy efficiency.

Other Ring-based Topologies: Spidergon [Coppola et al. 2004] proposes a bidirectional ring augmented with links that directly connect nodes opposite each other on the ring. These additional links reduce the average hop distance for traffic. However, the cross-ring links become very long as the ring grows, preventing scaling past a certain point, whereas our design has no such scaling bottleneck. Octagon [Karim et al. 2001] forms a network by joining Spidergon units of 8 nodes each. Units are joined by sharing a "bridge node" in common. Such a design scales linearly. However, it it does not make use of hierarchy, while our design makes use of global rings to join local rings.

Other Low Cost Router Designs: Kim [Kim 2009] proposes a low-cost router design that is superficially similar to our proposed node router design: routers convey traffic along rows and columns in a *mesh* without making use of crossbars, only pipeline registers and MUXes. Once traffic enters a row or column, it continues until it reaches its destination, as in a ring. Traffic also transfers from a row to a column analogously to a ring transfer in our design, using a "turn buffer." However, this design is explicitly designed for meshes, hence would not be directly usable in our ring-based design because of potential livelock as we discussed in Section 4 and this design will have to provide an additional delivery guarantee mechanism (as we proposed in Section 4). Additionally, this design does not use deflections when there is a contention. Mullins et al. [Mullins et al. 2004] propose a buffered mesh router with single-cycle arbitration. We share the goal of building a simpler, faster router. Our work differs in that our focus is on hierarchical rings rather than meshes. Abad et al. [Abad et al. 2007] proposed a router design, called the Rotary Router, that consists of two independent rings that join the router's ports and perform packet arbitration similar to standalone ring-based networks. Both the Rotary Router and HiRD allow a packet to circle a ring again in a "deflection" if an ejection (ring transfer or router exit) is unsuccessful. However, their design fundamentally differs from ours because each router has an internal ring, and the network as a whole is a mesh. In contrast, HiRD's routers are simpler as they are designed for hierarchical ring topology. Kodi et al. [Kodi et al. 2008] propose an orthogonal mechanism that reduces buffering by using dual-function links as buffer space when necessary.

Bufferless Mesh-based Interconnects: While we focus on ring-based interconnects to achieve simpler router design and lower power, other work modifies conventional buffered mesh routers by removing the buffers and using deflection [Baran 1964; Gómez et al. 2008b; Hayenga et al. 2009; Konstantinidou and Snyder 1991; Moscibroda and Mutlu 2009; Fallin et al. 2011]. As we show in our evaluations for CHIPPER [Fallin et al. 2011], while such designs successfully reduce power and area, they retain the fundamental complexity of mesh-based routers. Applying bufferless routing principles to rings leads to inherently simpler designs, as there is only one option for deflection in a ring (i.e., continue circulating around the ring). Other works propose dropping packets under contention [Gómez et al. 2008b; 2008a] and SCARAB [Hayenga et al. 2009] adds a dedicated circuit-switch network to send retransmit requests. Several machines such as HEP [Smith 1981], Tera [Alverson et al. 1990] and the Connection Machine [Hillis 1989] also use deflection routing to connect different chips.

8. CONCLUSION

We introduced *HiRD*, for *Hierarchical Rings with Deflection*, which is a simple hierarchical ring-based NoC design. Past work has shown that a hierarchical ring design yields good performance and scalability relative to both a single ring and a mesh. HiRD has two new contributions: (1) a simple router design that enables ring transfers *without in-ring buffering or flow control*, instead using limited *deflections* (retries) when a flit cannot transfer to a new ring, and (2) two *guarantee mechanisms* that ensure deterministically guaranteed forward progress despite deflections. Our evaluations show that HiRD enables a simpler and lower-cost implementation of a hierarchical ring network. Although an exhaustive topology comparison is not the goal of this work, our evaluations also show that HiRD is competitive with a variety of other baseline designs in both performance and energy efficiency. We conclude that HiRD represents a compelling interconnect design point to bring additional scalability to existing ring-based designs at high energy efficiency.

REFERENCES

Pablo Abad, Valentin Puente, Valentin Puente, and Pablo Prieto. 2007. Rotary router: an efficient architecture for CMP interconnection networks. *ISCA* (2007).

Robert Alverson, David Callahan, Daniel Cummings, Brian Koblenz, Allan Porterfield, and Burton Smith. 1990. The Tera computer system. (1990).

P. Baran. 1964. On distributed communications networks. *IEEE Trans. on Comm.* (1964).

E Baydal, P Lopez, and J Duato. 2005. A family of mechanisms for congestion control in wormhole networks. *IEEE Trans. on Par. and Dist. Sys.* 16 (2005). Issue 9.

C Carrión, R. Beivide, J.A. Gregorio, and F. Vallejo. 1997. A flow control mechanism to avoid message deadlock in k-ary n-cube networks. *High Performance Computing* (1997).

Kevin Kai-Wei Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu. 2012. HAT: Heterogeneous Adaptive Throttling for On-Chip Networks. *Computer Architecture and High Performance Computing, Symposium on* (2012).

M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. 2004. Spidergon: a novel on-chip communication network. *Proc. Int'l Symposium on System on Chip* (Nov 2004).

David E. Culler, J.P. Singh, and Anoop Gupta. 1999. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann.

W Dally. 1992. Virtual-channel flow control. *IEEE Par. and Dist. Sys.* (1992).

W Dally and C Seitz. 1987. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. on Comp.* (1987).

W Dally and B Towles. 2004. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann.

W. J. Dally and B Towles. 2001. Route packets, not wires: On-chip interconnection networks. *DAC-38* (2001).

R Das, S. Eachempati, A.K. Mishra, V. Narayanan, and C.R. Das. 2009. Design and Evaluation of Hierarchical On-Chip Network Topologies for next generation CMPs. *HPCA-15* (2009).

Thomas H. Dunigan. 1994. Kendall Square Multiprocessor: Early Experiences and Performance. In *of the Intel Paragon, ORNL/TM-12194*.

S Eyerman and L Eeckhout. 2008. System-Level performance metrics for multiprogram workloads. *IEEE Micro* 28 (May 2008), 42–53. Issue 3.

C Fallin, C Craik, and O Mutlu. 2011. CHIPPER: A Low-complexity bufferless deflection router. *HPCA-17* (2011).

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu. 2012. MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect. *NOCS 2012* (2012).

C Gómez, M Gómez, P López, and J Duato. 2008a. A Bufferless switching technique for NoCs. *Wina* (2008).

C Gómez, M Gómez, P López, and J Duato. 2008b. Reducing packet dropping in a bufferless NoC. *EuroPar-14* (2008).

R Grindley, T. Abdelrahman, S. Brown, S. Caranci, D. DeVries, B. Gamsa, A. Grbic, M. Gusat, R. Ho, O. Krieger, G. Lemieux, K. Loveless, N. Manjikian, P. McHardy, S. Srbljic, M. Stumm, Z. Vranesic, and Z. Zilic. 2000. The NUMAchine multiprocessor. *In Proc. 29th Int'l Conf. on Parallel Processing* (2000), 487–496.

D Gustavson. 1992. The Scalable Coherent Interface and Related Standards Projects. *IEEE Micro* 12 (Feb. 1992), 10 – 22. Issue 1.

V. Carl Harmacher and Hong Jiang. 2001. Hierarchical Ring Network Configuration and performance Modeling. *IEEE Transaction on Computers* 50, 1 (2001), 1–12.

M Hayenga, N Enright Jerger, and M Lipasti. 2009. SCARAB: A Single Cycle Adaptive Routing and Bufferless Network. *MICRO-42* (2009).

John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* (2006).

W.D. Hillis. 1989. *The Connection Machine*. MIT Press.

Intel Corporation. 2011. Intel Details 2011 Processor Features. http://newsroom.intel.com/community/intel_newsroom/blog/2010/09/13/intel-details-2011-processor-features-offers-stunning-visuals-built-in. (2011).

F Karim, A Nguyen, S Dey, and R Rao. 2001. On-Chip Communication Architecture for OC-768 Network Processors. *DAC-38* (2001).

J Kim. 2009. Low-Cost Router Microarchitecture for On-Chip Networks. *MICRO-42* (2009).

J Kim and W Dally. 2007. Flattened butterfly: A cost-efficient topology for high-radix networks. *ISCA-34* (2007).

John Kim and Hanjoon Kim. 2009. Router microarchitecture and scalability of ring topology in on-chip networks. *NoCArc* (2009).

A Kodi, A Sarathy, and A Louri. 2008. iDEAL: Inter-router dual-function energy and area-efficient links for network-on-chip (NoC) architectures. *ISCA-35* (2008).

S. Konstantinidou and L. Snyder. 1991. Chaos router: architecture and performance. *ISCA-18* (1991).

David Kroft. 1981. Lockup-free instruction fetch/prefetch cache organization. *ISCA-8* (1981).

Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media? *WWW* (2010).

Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. *OSDI* (2012).

J Laudon and D Lenoski. 1997. The SGI Origin: a ccNUMA Highly Scalable Server. *ISCA-24* (1997).

Hyunjin Lee, Sangyeun Cho, and B.R. Childers. 2011. CloudCache: Expanding and shrinking private caches. *HPCA-17* (2011).

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. GraphLab: A New Parallel Framework for Machine Learning. *UAI* (2010).

Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. *PLDI* (2005).

George Michelogiannakis, D Sanchez, WJ Dally, and C Kozyrakis. 2010. Evaluating bufferless flow-control for on-chip networks. *NOCS* (2010).

Thomas Moscibroda and Onur Mutlu. 2009. A Case for Bufferless Routing in On-Chip Networks. *ISCA-36* (2009).

Robert Mullins, Andrew West, and Simon Moore. 2004. Low-latency virtual-channel routers for on-chip networks. *ISCA-31* (2004).

George Nychis, Chris Fallin, Thomas Moscibroda, and Onur Mutlu. 2010. Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need? *Hotnets-IX* (2010).

D.C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P.M. Harvey, H.P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D.L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. 2006. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *J. Solid-State Circuits* 41, 1 (Jan 2006), 179–196.

G Ravindran and M Stumm. 1997. A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks. *HPCA-3* (1997).

G Ravindran and M Stumm. 1998. On topology and bisection bandwidth for hierarchical-ring networks for shared memory multiprocessors. *HPCA-4* (1998).

Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. 2008. Larrabee: a many-core x86 architecture for visual computing. *SIGGRAPH* (2008).

B Smith. 1981. Architecture and applications of the HEP multiprocessor computer system. *SPIE* (1981).

Allan Snavely and Dean M. Tullsen. 2000. Symbiotic jobscheduling for a simultaneous multithreaded processor. *ASPLOS-9* (2000).

Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. *NOCS* (2012).

M Thottethodi, AR Lebeck, and S Mukherjee. 2001. Self-tuned congestion control for multiprocessor networks. *HPCA-7* (2001).

Sergio Tota, Mario R. Casu, and Luca Macchiarulo. 2006. Implementation analysis of NoC: a MPSoC trace-driven approach. *GLSVLSI-16* (2006).

Aniruddha N. Udipi, Naveen Muralimanohar, and Rajeev Balasubramonian. 2010. Towards scalable, energy-efficient, bus-based on-chip networks. *HPCA-16* (2010).

H Wang, L Peh, and S Malik. 2003. Power-driven design of router microarchitectures in on-chip networks. *MICRO-36* (2003).

X Zhang and Y Yan. 1995. Comparative Modeling and Evaluation of CC-NUMA and COMA on Hierarchical Ring Architectures. *Parallel and Distributed Systems, IEEE Transactions on* 6, 12 (Dec 1995), 1316 – 1331.