2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing

Design and Evaluation of Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun Chris Fallin Xiangyao Yu† Kevin Kai-Wei Chang
Greg Nazario Reetuparna Das§ Gabriel H. Loh‡ Onur Mutlu

Carnegie Mellon University §University of Michigan †MIT ‡Advanced Micro Devices, Inc.

***Abstract*—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.

I. 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 [11]: cores exchange packets over a network consisting of network switches and links arranged in some topology.

Mainstream commercial CMPs today most commonly use *ring*-based interconnects. Rings are a well-known network topology [10], 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 [45], IBM Cell [42], and more recently, the Intel Sandy Bridge [24].

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.

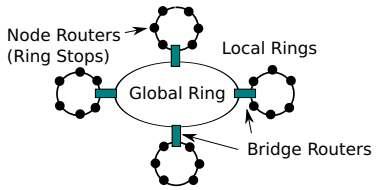


Fig. 1: A traditional hierarchical ring design [43, 51, 21, 44, 19] allows "local rings" with simple node routers to scale by connecting to a "global ring" via bridge routers.

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 [43, 51, 21, 44, 19] 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 [43]. The state-of-the-art design [43] 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 much more scalable than a single ring [43], 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 [4, 23, 46, 2, 38, 17], 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 §III and evaluate with worst-case traffic in §V-C).

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 buffered hierarchical ring design [43].

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

1550-6533/14 $31.00 © 2014 IEEE
DOI 10.1109/SBAC-PAD.2014.31

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 hierarchical ring design with in-ring buffering and buffered flow control [43]. 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.

II. 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 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 §III.

While deflection-based bufferless routing has been previously proposed and evaluated for a variety of off-chip and on-chip interconnection networks (e.g., [4, 38, 17, 15, 16, 40, 41]), deflections are trivially implementable in a ring: if deflection occurs, the flit[1] 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 outbound ports, leading to much more circuit complexity in the router microarchitecture, as shown by [15, 22, 37]. Our application of deflection to rings leads to a simple and elegant embodiment of bufferless routing.

A. Node Router Operation

At each node on a local ring, we place a single node router, shown in Figure 2. 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 is very common in ring-based and ring-like designs (e.g., [28]).

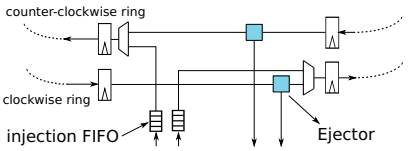


Fig. 2: Node router.

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.[2] Note that the flits constituting a packet may arrive out-of-order and at widely separated times. Reassembly 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 [15]. With this scheme, receivers reassemble packets in-place in MSHRs (Miss-Status Handling Registers [33]), 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 [15]. Furthermore, it ensures that a ring full of flits always drains, thus ensuring forward progress (as we will describe more fully in §III).

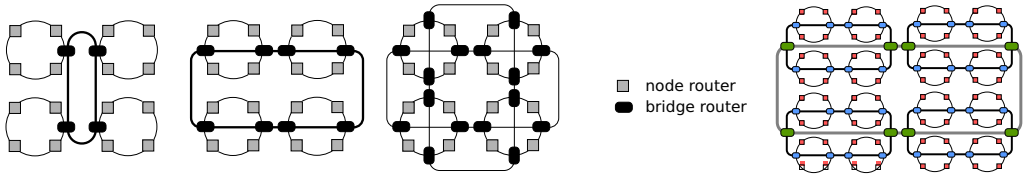
After locally-destined traffic is removed from the ring, the remaining traffic travels to the injection stage. At this stage, the router looks for "empty slots," or cycles where no flit is present on the ring, and injects new flits into the ring whenever they are queued for injection. The injector is 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.

B. Bridge Routers

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 §II-C), 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 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 [4], also known as deflection routing, and has been used in recent on-chip mesh interconnect designs to resolve contention [38, 15, 16, 49, 40, 41]. Note that to ensure that flits are *eventually* delivered, despite any deflections that may occur, we introduce two *guarantee mechanisms*

[1] All operations in the network happen in a flit level similar to previous works [38, 17, 15, 16, 40, 41].

[2] 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 baselines.



(a) 4-, 8-, and 16-bridge hierarchical ring designs. **(b)** Three-level hierarchy (8x8).

Fig. 3: Hierarchical ring design of HiRD.

in §III. 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 [15], packet reassembly works despite out-of-order arrival.

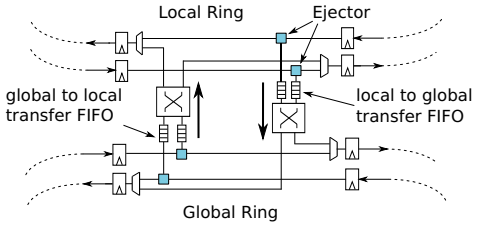


Fig. 4: Bridge router.

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* [23]) 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. Figure 3a shows three designs denoted by the number of bridge routers in total: 4-bridge, 8-bridge, and 16-bridge. 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. We use a 3-level hierarchy, illustrated in Figure 3b, to build a 64-node network.

Finally, in order to address a potential deadlock case (which will be explained more in §III), 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). It 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.

C. 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 more local levels), it takes any bridge router which connects to the next level, until it finally reaches the local ring of its destination and ejects at the node with a full address match.

III. 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.

To understand the need for each guarantee, let us consider an example, shown in Figure 5. 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 ① to 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 ①, the bridge router B1 ②, the global ring ③, the bridge router B2 ④, the rightmost local ring ⑤, and finally the destination node N2 ⑥.

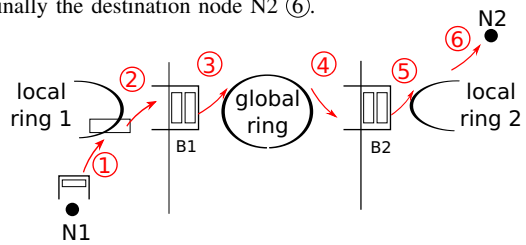


Fig. 5: The need for the injection and transfer guarantees: contention experienced by a flit during its journey.

In the worst case, when the network is heavily contended, the flit could wait for an unbounded amount of time at ① to 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 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 that 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 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 (§III-A) and the transfer guarantee (§III-B) mechanisms. We show correctness in §III-C, and quantitatively evaluate both mechanisms in §V-C and in [3].

A. 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 de-asserts its throttling request signal to the global controller, and the global controller subsequently allows all other nodes to resume normal operation.

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.[3] if any node in the ring still cannot inject, the bridge routers connected to that ring start sending throttling signals to any other ring in the next level of the ring hierarchy they are 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 requires two wires in each ring and small design overhead at the bridge router to propagate the throttling signal across hierarchy levels. In our evaluation, we faithfully model this hierarchical design.

B. 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 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.

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

C. Putting it Together: Guaranteed Delivery

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 mechanism [15], as already described in §II-A). 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 any node 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 deadlock 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. The transfer guarantee does exactly this by observing ring slots in sequence and reserving a transfer queue entry 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).

Parameter	Setting
System topology	CPU core and shared cache slice at every node
Core model	Out-of-order, 128-entry ROB, 16 MSHRs (maximum 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
Cache coherence	Directory-based protocol (based on SGI Origin [34]), directory entries co-located with shared cache blocks
Simulation length	5M-instruction warm-up, 25M-instruction active execution per node [38, 15, 6, 16]

TABLE I: 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 (link+router); 64-bit local and 128-bit global rings, **8x8:** three-level hierarchy, 4x4 parameters, with second-level rings connected by a 256-bit third-level ring
	HiRD	**4x4:** 2-cycle (local), 3-cycle (global) per-hop latency (link+router); 64-bit local ring, 128-bit global ring; **8x8:** 4x4 parameters, with second-level rings connected by a 256-bit third-level ring
Router	Single Ring	1-cycle per-hop latency (as in [30])
	Buffered HRing	Node (NIC) and bridge (IRI) routers based on [43]; 4-flit in-ring and transfer FIFOs. Bidirectional links of dual-flit width (for fair comparison with our design). Bubble flow control [5] for deadlock freedom.
	HiRD	Local-to-global buffer depth of 1, global-to-local buffer depth of 4

TABLE II: Network parameters.

D. Hardware Cost

Our injection and transfer guarantee mechanisms have 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 (§III-A). 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 evaluated 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).

IV. Evaluation Methodology

We perform our evaluations using a cycle-accurate simulator of a CMP system with 1.6GHz interconnect to provide application-level performance results. Details are given in Tables I and II. Our methodology ensures a rigorous and isolated evaluation of NoC capacity for especially cache-resident workloads, and has also been used in other studies [38, 15, 40, 41, 16]. Instruction traces for the simulator are taken using a Pintool [36] on representative portions of SPEC CPU2006 workloads.

We mainly compare to a single bidirectional ring and a state-of-the-art buffered hierarchical ring [43]. Also, note that while there are many possible ways to optimize each baseline (such as congestion control [6, 40, 41], adaptive routing schemes, and careful parameter tuning), we assume a fairly typical aggressive configuration for each.

Data Mapping: We map data in a cache-block-interleaved way to different shared L2 cache slices. 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 as in [35, 8]. However, such a mapping mechanism is orthogonal to our proposal and can be applied in all ring-based network designs.

Application & Synthetic Workloads: 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 (H), Medium (M), Medium-Low (ML), and Low (L), 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).

Energy & Area: We measure the energy and area of routers and links 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 [48]. Control logic is modeled in Verilog RTL. 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 6.

Application Evaluation Metrics: For multiprogrammed workloads, we present application performance results using the commonly-used Weighted Speedup metric [47, 14].

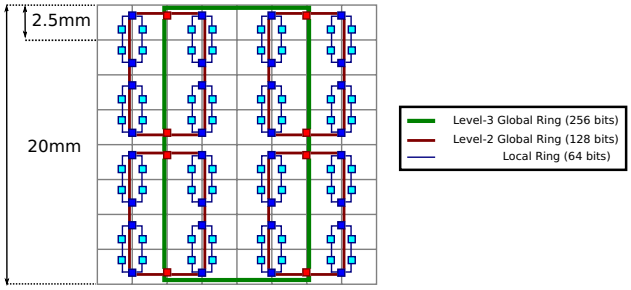


Fig. 6: Assumed floorplan for HiRD 3-level (64-node) network. Two-level (16-node) network consists of one quadrant of this floorplan.

V. Evaluation

We provide an evaluation of our proposed mechanism against other ring baselines. Since our goal is to provide a better ring design, our main comparisons are to ring networks. We have many additional results and analyses, which we cannot present due to space limitations. These are provided in our technical report [3]. Some of these results show our mechanism works well with multithreaded workloads and provides better worst-case latencies than the baselines. Additional results also provide sensitivity analyses with different system and design parameters. See [3] for details.

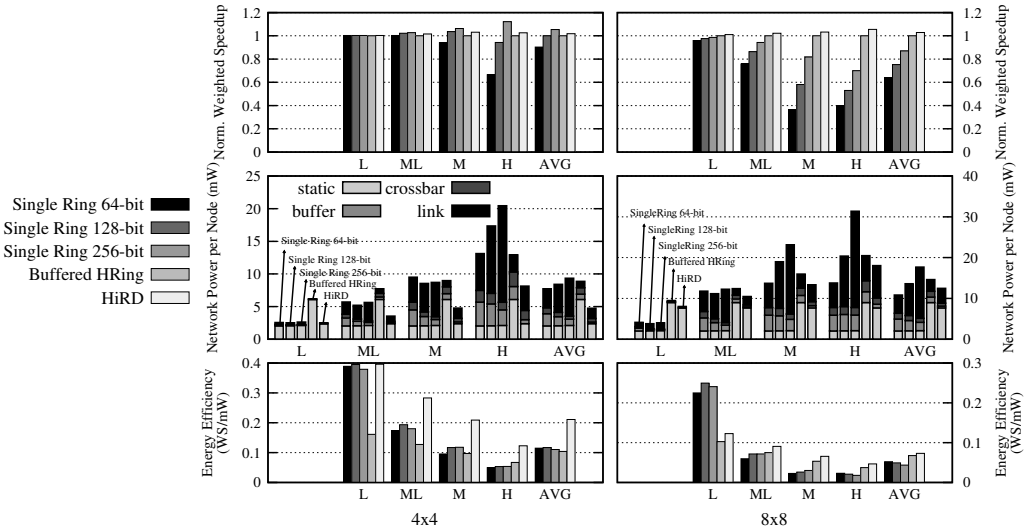


Fig. 7: HiRD as compared to buffered hierarchical rings and a single-ring network.

A. Ring-based Network Designs

Figure 7 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 and buffered hierarchical rings in [43], using identical topologies, as well as a single ring (with different bisection bandwidths).

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 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% higher performance.

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

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 higher dynamic energy due to 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.

Our technical report [3] contains more analyses and additional evaluations on several multithreaded workloads. Our results show that HiRD is more energy-efficient than both the single ring designs and the buffered hierarchical ring design.

B. Synthetic-Traffic Network Behavior

Figure 8 shows the average packet 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 [10]. 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 256-bit single ring saturates at a lower injection rate. As network

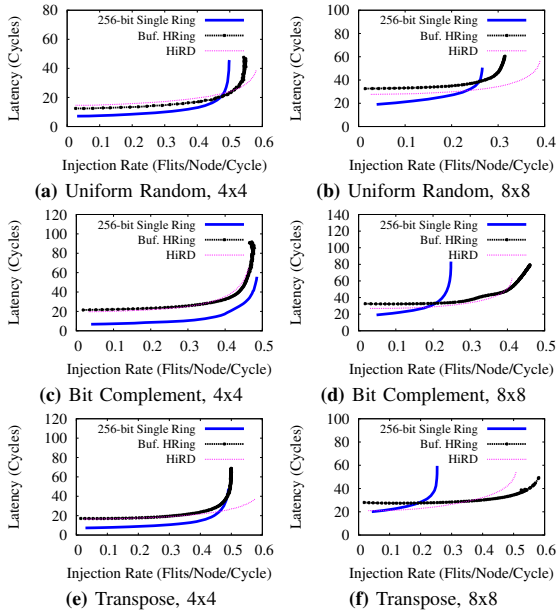


Fig. 8: Synthetic-traffic evaluations for 4x4 and 8x8 networks.

size scales to 8x8, HiRD performs significantly better than the 256-bit single ring, 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 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 (§III) 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 at a higher injection rate than the buffered hierarchical ring design.

C. 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.

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

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

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 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 §III intercedes).

Results: Table III 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 (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 guarantees, 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 throughput. Thus, the guarantees are both necessary and sufficient to ensure deterministic forward progress for all flits in the network.

Our technical report [3] provides similar worst-case analyses for real workloads as well as additional network latency results. We have shown in [3] that the injection and transfer guarantees also lower network latency while deflections help to balance network load under heavy traffic.

D. Router Area and Timing

We show both critical path length and normalized die area for single-ring, buffered hierarchical ring, and HiRD, in Table IV. 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).

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

TABLE IV: Total router area (16-node network) and critical path.

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% vs. the buffered hierarchical 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).

E. Comparison Against Other Ring Configurations

Figure 9 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 §II-B). 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.

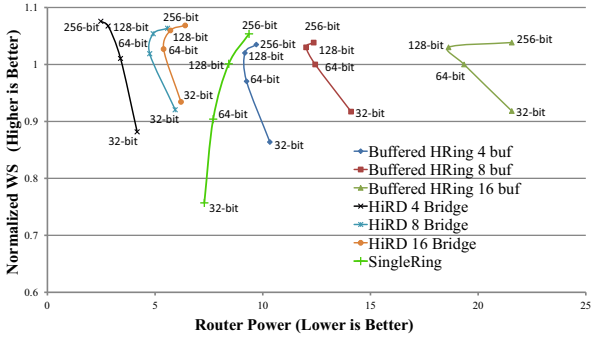


Fig. 9: Weighted speedup (Y) vs. power (X) for 4x4 networks.

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: 1) links and crossbars consume more energy, 2) packets arrive at the destination faster, leading to higher power as more energy is consumed in less time.

Aside from the comparison against other ring configurations, comparisons against other network topologies such as buffered [9] and bufferless [15] meshes and flattened butterfly [29], are available in our technical report [3]. We show in [3] that HiRD is more energy efficient than these network topologies.

VI. 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 within rings while guaranteeing end-to-end packet delivery.

Hierarchical Interconnects: Hierarchical ring-based interconnect was proposed in a previous line of work [43, 51, 21, 44, 19, 27]. 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, wormhole switching and flow control. Concurrent works by Kim et al. propose tNoCs, hybrid packet-flit credit-based flow control [27] and Clumsy Flow Control (CFC) [26]. However, these two designs add additional complexity because tNoCs [27] requires an additional credit network to guarantee forward progress while CFC requires coordination between cores and memory controllers. In contrast, flow control in HiRD is lightweight (with deflection based flow control, the Retransmit-Once mechanism, and simpler local-to-global and global-to-local buffers). Additionally, throttling decisions in HiRD can be made locally in each local ring as opposed to global decisions in CFC [26] and tNoCs [27].

Udipi et al. proposed a hierarchical topology using global and local buses [50]. 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. [12] examined several hierarchical designs, including a concentrated mesh (one mesh router shared by several nearby nodes).

A previous system, SCI (Scalable Coherent Interface) [20], 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 a ring), 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 [13] 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 [7] 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 [25] 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 [28] proposes a low-cost router design. 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 §III. Additionally, this design does not use deflections when there is contention. Mullins et al. [39] propose a buffered mesh router with single-cycle arbitration. Our work differs in that our focus is on hierarchical rings rather than meshes. Abad et al. [1] propose the Rotary Router. 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 rings. Kodi et al. [31] propose an orthogonal mechanism that reduces buffering by using 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 buffers and using deflection [4, 18, 22, 32, 38, 15, 6, 16, 40, 41]. 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 [18, 17]. SCARAB [22] adds a dedicated circuit-switch network to send retransmit requests. Several machines such as HEP [46], Tera [2] and the Connection Machine [23] also use deflection routing to connect different chips.

VII. Conclusion

We introduced *HiRD*, 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 [3] also show that HiRD is more energy-efficient than several other topologies while providing competitive performance. We conclude that HiRD represents a compelling interconnect design point to bring additional scalability to existing ring-based designs at high energy efficiency.

Acknowledgments

We thank the reviewers and SAFARI members for their feedback. We acknowledge the support of AMD, IBM, Intel, and Qualcomm. This research was partially supported by ISTC-CC, NSF (CCF 0953246 and CCF 1212962), and SRC. Rachata Ausavarungnirun is supported by the Royal Thai Government scholarship.

References

[1] P. Abad et al. Rotary router: an efficient architecture for CMP interconnection networks. *ISCA*, 2007.
[2] R. Alverson et al. The Tera computer system. *ICS*, 1990.
[3] R. Ausavarungnirun et al. Improving energy efficiency of hierarchical rings via deflection routing. SAFARI Technical Report TR-2014-002: http://safari.ece.cmu.edu/tr.html, Apr 2014.
[4] P. Baran. On distributed communications networks. *IEEE Trans. on Comm.*, 1964.
[5] C. Carrión et al. A flow control mechanism to avoid message deadlock in k-ary n-cube networks. *HiPC*, 1997.
[6] K. K.-W. Chang et al. HAT: Heterogeneous adaptive throttling for on-chip networks. *SBAC-PAD*, 2012.
[7] M. Coppola et al. Spidergon: a novel on-chip communication network. *SOCC*, Nov 2004.
[8] C. Craik and O. Mutlu. Investigating the viability of bufferless NoCs in modern chip multi-processor systems. SAFARI Technical Report TR-2011-004: http://safari.ece.cmu.edu/tr.html, Aug 2011.
[9] W. Dally. Virtual-channel flow control. *IEEE TPDS*, 1992.
[10] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.
[11] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *DAC*, 2001.
[12] R. Das et al. Design and evaluation of hierarchical on-chip network topologies for next generation CMPs. *HPCA*, 2009.
[13] T. H. Dunigan. Kendall square multiprocessor: Early experiences and performance. In *of the Intel Paragon, ORNL/TM-12194*, 1994.
[14] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 2008.
[15] C. Fallin et al. CHIPPER: A low-complexity bufferless deflection router. *HPCA*, 2011.
[16] C. Fallin et al. MinBD: Minimally-buffered deflection routing for energy-efficient interconnect. *NOCS*, 2012.
[17] C. Gómez et al. BPS: a bufferless switching technique for NoCs. *Wina*, 2008.
[18] C. Gómez et al. Reducing packet dropping in a bufferless NoC. *EuroPar*, 2008.
[19] R. Grindley et al. The NUMAchine multiprocessor. *ICPP*, 2000.
[20] D. Gustavson. The scalable coherent interface and related standards projects. *IEEE Micro*, 1992.
[21] V. C. Hamacher and H. Jiang. Hierarchical ring network configuration and performance modeling. *IEEE Transaction on Computers*, 2001.
[22] M. Hayenga et al. Scarab: A single cycle adaptive routing and bufferless network. *MICRO*, 2009.
[23] W. Hillis. *The Connection Machine*. MIT Press, 1989.
[24] Intel Corporation. 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.
[25] F. Karim et al. On-chip communication architecture for OC-768 network processors. *DAC*, 2001.
[26] H. Kim et al. Clumsy flow control for high-throughput bufferless on-chip networks. *IEEE CAL*, 2013.
[27] H. Kim et al. Transportation-network-inspired network-on-chip. *HPCA*, 2014.
[28] J. Kim. Low-cost router microarchitecture for on-chip networks. *MICRO*, 2009.
[29] J. Kim and W. Dally. Flattened butterfly: A cost-efficient topology for high-radix networks. *ISCA*, 2007.
[30] J. Kim and H. Kim. Router microarchitecture and scalability of ring topology in on-chip networks. *NoCArc*, 2009.
[31] A. Kodi et al. iDEAL: Inter-router dual-function energy and area-efficient links for network-on-chip (NoC) architectures. *ISCA*, 2008.
[32] S. Konstantinidou and L. Snyder. Chaos router: architecture and performance. *ISCA*, 1991.
[33] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. *ISCA*, 1981.
[34] J. Laudon and D. Lenoski. The SGI Origin: a ccNUMA highly scalable server. *ISCA*, 1997.
[35] H. Lee et al. Cloudcache: Expanding and shrinking private caches. *HPCA*, 2011.
[36] C.-K. Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. *PLDI*, 2005.
[37] G. Michelogiannakis et al. Evaluating bufferless flow-control for on-chip networks. *NOCS*, 2010.
[38] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. *ISCA*, 2009.
[39] R. Mullins et al. Low-latency virtual-channel routers for on-chip networks. *ISCA*, 2004.
[40] G. P. Nychis et al. Next generation on-chip networks: What kind of congestion control do we need? *Hotnets*, 2010.
[41] G. P. Nychis et al. On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects. *SIGCOMM*, 2012.
[42] D. Pham et al. Overview of the architecture, circuit design, and physical implementation of a first-generation CELL processor. *JSSC*, 2006.
[43] G. Ravindran and M. Stumm. A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks. *HPCA*, 1997.
[44] G. Ravindran and M. Stumm. On topology and bisection bandwidth for hierarchical-ring networks for shared memory multiprocessors. *HPCA*, 1998.
[45] L. Seiler et al. Larrabee: a many-core x86 architecture for visual computing. *SIGGRAPH*, 2008.
[46] B. Smith. Architecture and applications of the HEP multiprocessor computer system. *SPIE*, 1981.
[47] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. *ASPLOS*, 2000.
[48] C. Sun et al. DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. *NOCS*, 2012.
[49] S. Tota et al. Implementation analysis of NoC: a MPSoC trace-driven approach. *GLSVLSI*, 2006.
[50] A. N. Udipi et al. Towards scalable, energy-efficient, bus-based on-chip networks. *HPCA*, 2010.
[51] X. Zhang and Y. Yan. Comparative modeling and evaluation of CC-NUMA and COMA on hierarchical ring architectures. *IEEE TPDS*, 1995.