HiRD: A Low-Complexity, Energy-Efficient Hierarchical Ring Interconnect

Chris Fallin
cfallin@cmu.edu
Xiangyao Yu†
yxy@mit.edu
Kevin Chang
kevincha@cmu.edu
Rachata Ausavarungnirun
rausavar@cmu.edu
Greg Nazario
gnazario@cmu.edu
Reetuparna Das§
reetudas@umich.edu
Onur Mutlu
onur@cmu.edu

Computer Architecture Lab (CALCM)
Carnegie Mellon University

†Massachusetts Institute of Technology
§University of Michigan

SAFARI Technical Report No. 2012-004

December 13, 2012

Abstract

Energy consumption and design simplicity are paramount concerns in on-chip interconnects for chip multiprocessors. Several proposed and a few implemented many-core on-chip interconnects are mesh or torus-based. These designs offer good scalability. However, most mainstream commercial chip multiprocessors use rings, in which each network node has relatively simpler ring stop logic. Network traffic injected into the ring continues until reaching its destination, so no flow control or buffering is needed, unlike a mesh. This design simplicity is attractive to implementors of small-to-medium-scale CMPs, and at lower core counts, rings can offer competitive performance with lower die area and energy consumption. Unfortunately, rings do not scale as well as meshes to large core counts.

In this paper, we propose a simple *hierarchical ring* topology and router design, which we call **HiRD** (Hierarchical Rings with Deflection), to enable better scalability while maintaining the simplicity of existing ring-based designs. Hierarchical ring networks have been proposed before. However, HiRD has two major new contributions. *First*, unlike past hierarchical ring designs, HiRD requires no in-ring flow control or buffering. Instead, HiRD implements inter-ring transfers using "bridge routers" which use minimal inter-ring buffering and, when the buffer is full, *deflect* transferring flits so that they circle the ring and try again. *Second*, we introduce two simple mechanisms which provide an end-to-end delivery guarantee (despite any deflections that occur) without impacting the critical path or latency of the vast majority of network traffic. We rigorously show that our network is deadlock- and livelock-free.

Our evaluations show that HiRD attains equal or better performance at better energy efficiency than a comprehensive set of baseline NoC topologies and router designs, including a previous hierarchical ring design, a conventional 2D mesh, and a single ring. We conclude that HiRD is a compelling design point which allows scalable, efficient performance while retaining the simplicity and appeal of ring-based 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 busses) are no longer scalable. The Network-on-Chip (NoC) is the most commonly-proposed solution [7]: cores exchange packets over a network consisting of network switches and links arranged in some topology.

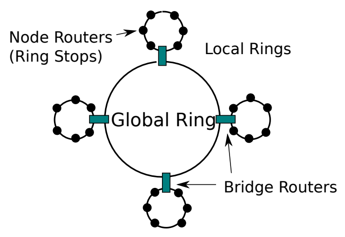


Figure 1: A hierarchical ring topology allows "local rings" with simple node routers to scale by connecting to a "global ring" via bridge routers.

To achieve scalability to many (tens, hundreds, or thousands) of cores, *mesh* or other higher-radix topologies are used. Both the Intel SCC [2] and Terascale [19] CMPs, and several other many-core CMPs (e.g., the MIT RAW [38] prototype, the UT-Austin TRIPS chip [13], and several Tilera products [44, 39]), exchange packets on a mesh. A mesh topology has good scalability because there is no central structure which needs to scale (unlike a central bus or crossbar-based design), and bisection bandwidth increases as the network grows. However, routers in a 2D mesh can consume significant energy and die area because of overheads in buffering, in routing and flow control, and in the switching elements (crossbars) which connect multiple inputs with multiple outputs. For example, in the Intel Terascale chip, the mesh interconnect consumes 30% of chip power [19]. More recently, the Intel 48-core SCC reported 10% of chip power spent in interconnect [2]. Previous work has addressed both power and area by eliminating router buffers [40, 12, 30, 18, 10] (and performing deflection routing [1, 26]) or using a simplified microarchitecture [23], or reducing network power by switching off buffers or links [21, 16]. Nevertheless, mesh-based designs have fundamental overheads not present in ring routers, and the complexity of mesh routers is a deterrent for commercial adoption because of the design and validation effort required.

In contrast to these mesh-based manycore systems, mainstream commercial CMPs today most commonly use *ring*-based interconnects. Rings are a well-known network topology [6], 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 crossbar; 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 decision or output port allocation is 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 [35], IBM Cell [32], and more recently, the Intel Sandy Bridge [20].

Past work has shown that rings are competitive with meshes up to tens of nodes [25]. 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 which 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. Fig. 1 shows an example of such a *hierarchical ring* topology. Past work [33, 45, 17, 34, 14] has proposed hierarchical rings as a scalable alternative to single ring and mesh networks. These proposals join rings with *bridge routers*, which reside on multiple rings and transfer traffic between rings. This topology is shown to yield good performance and scalability [33]. However, the state-of-the-art design [33] 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. We observe that although the hierarchical ring topology shows promise, the past design's requirement for flow control and buffering adds die area and energy cost and increases design and validation effort, especially for existing systems which currently use single rings (without flow control or buffering).

Our goal is to build a hierarchical ring network in a simple way, without the need for in-ring flow control or buffering. Such a design would maintain the appeal of existing, very simple, ring routers, while achieving the scalability that past work has shown for the hierarchical topology.[1] In this work, we propose such a design based on a hierarchy of rings. The local rings operate as before, injecting new packets whenever there is a free slot and removing packets when they reach their destinations. These local rings are joined by one or more global rings. In order to facilitate transfers between local and global rings, we design a new *bridge router* which accepts packets from one ring, places them in a transfer FIFO, and injects them into the other ring when a free slot is available.

Our key idea is to allow a bridge router to *deflect* packets if the destination ring is busy and the transfer FIFO is full. This design choice differs from past work, which required in-ring flow control such that one ring might stall when another ring is congested and a transfer is required. In HiRD, when a ring transfer cannot be made and a *deflection* occurs, the deflected traffic makes another round-trip on its current ring and tries again when it next reaches a bridge router. This retry-based scheme avoids the need for buffering and flow control within any ring. Finally, to ensure forward progress, we introduce two new mechanisms, the *injection guarantee* and the *transfer guarantee*, which guarantee packet delivery in the worst case despite the best-effort retry-based transfer mechanism (as we discuss in §4 and evaluate with worst-case traffic in §7.3).

This simple hierarchical ring design, which we call *HiRD* (for Hierarchical Rings with Deflection), is a low-cost way to bring scalability to existing ring-based designs without the need for high-overhead buffering or flow control within the rings. We show in our evaluations that HiRD is competitive in performance and energy efficiency with respect to the buffered hierarchical ring design as well as single-ring and mesh-based designs. In summary, **our 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, and joins local and 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 topology and router designs. We show competitive performance to these baselines, with better energy efficiency than all prior designs, including the previously-proposed hierarchical ring with in-ring buffering [33]. We conclude that HiRD represents a compelling design point for future many-core interconnects by achieving scalable performance without sacrificing the simplicity of ring-based designs.

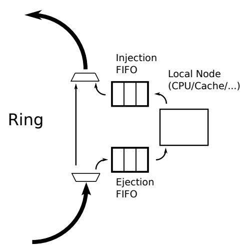
2 Motivation

In this section, we first argue that (i) ring routers can be very simple, compared to mesh routers, but that (ii) meshes are more scalable than single rings. We then describe past work in hierarchical rings and show how a previous design obtained performance scalability with rings but required in-ring flow control and buffering, which led to complicated ring routers. These two observations – the scalability of rings given hierarchy, but the complexity of prior approaches to hierarchical rings – motivate our *simple* hierarchical ring interconnect, HiRD.

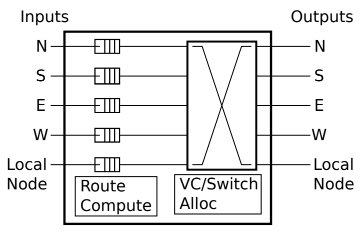
2.1 Simplicity in Ring-Based Interconnects

Ring interconnects are attractive in current small-to-medium-scale commercial CMPs because ring routers (ring stops) are simple, which leads to small 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. Fig. 2a 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.

[1]Note that our goal in this work is not to propose a new topology but to make the existing hierarchical ring topology simpler, more scalable and more energy-efficient, so that it can serve as a logical next step for designs which currently use single rings.



(a) A ring router.



(b) A mesh router.

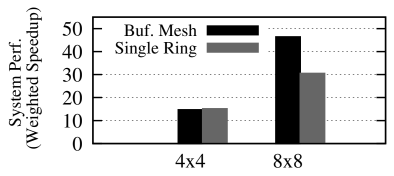


Figure 2: Performance as mesh and ring networks to 64 nodes.

In contrast, mesh-based interconnects, and other interconnects in which routers have higher *radix* (more connections to neighboring routers), require more complex routers. Fig. 2b depicts the router datapath and control logic necessary to implement a standard input-buffered (VC) mesh router [6]. For every incoming flit, the router stores the flit in a buffer, and also computes what the flit's next hop should be (*Route Computation*. This computation determines *which* output port the flit must take. Once this is known, the flit can arbitrate for a downstream buffer (*VC Arbitration*) and the datapath to the router output (*Switch Alloc*) before traversing the crossbar. All of these steps add energy and die area overhead that is not present in the simple ring-based router. Additionally, the design and validation effort required to correctly implement such a router is significantly higher than that for a simple ring router.

2.2 Scalability in Mesh-Based Interconnects

Despite the simplicity advantage of a ring-based network, rings have a fundamental *scalability* limit: as compared to a mesh, 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. (Intuitively, a packet visits half the network on its way to the destination, in the worst case, and a quarter in the average case, for a bidirectional ring.) In contrast, a mesh has bisection bandwidth proportional to one dimension of the layout (e.g., the square-root of the node count for a 2D mesh) and also has an average hop count that scales only with one dimension. The higher radix, and thus higher connectivity, of the mesh allows for more path diversity and lower hop counts which increases performance.

To demonstrate this problem quantitatively, we show application performance averaged over a representative set of network-intensive workloads on (i) a single ring network and (ii) a conventional 2D-mesh buffered interconnect (details in §6). As the plot shows, although the single ring is able to match the mesh's performance at the 16-node design point, it degrades when node count increases to 64. Note that in this case, the ring's bisection bandwidth is kept equivalent to the mesh, so the performance degradation is solely due to higher hop count; considerations of practical ring width might reduce performance further.

2.3 Hierarchical Rings for Scalability

So far, we have seen that (i) ring routers are very simple in comparison to mesh routers, but that (ii) a mesh offers better performance scalability at high node counts. Ideally, we would like a simple yet scalable interconnect.

Fortunately, past work has observed that *hierarchy* allows for additional scalability in many interconnects: in ring-based designs [33, 45, 17, 34, 14], with hierarchical buses [41], and with hierarchical meshes [8]. The hierarchical ring design [33] in particular reports promising results by combining local rings with one or more levels of global rings joined by Inter-Ring Interfaces (IRIs), which we call "bridge routers" in this work. Fig. 3 graphically depicts a detail of one bridge router in this buffered hierarchical ring network.

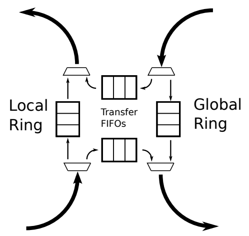


Figure 3: Buffered hierarchical ring detail, as proposed in prior work [33]: one *bridge router* which connects two rings.

However, 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. Clearly, a simpler way is needed to build bridge routers in a hierarchical ring network. We now introduce HiRD, which solves this problem in a low-cost way.

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 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.

3.1 Node Router Operation

At each node on a local ring, we place a single node router, shown in Fig. 4. 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, such as [23].

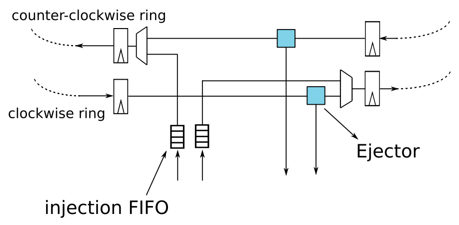


Figure 4: Node router.

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. 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 NoC [10]. With this scheme, receivers reassemble packets in-place in MSHRs (Miss-Status Handling Registers [27]), eliminating the need for separate reassembly buffers, and all traffic arriving at a node is consumed (or dropped) immediately, so ejection never places backpressure on the ring. Retransmit-Once also avoids protocol deadlock, because reassembly buffers are request buffers (MSHRs) and the same drop-based escape mechanism ensures forward progress.

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

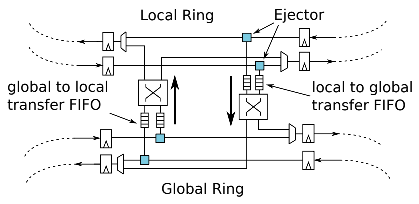


Figure 5: Bridge router.

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 Fig. 5. 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 §3.4, 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 [1], also known as deflection routing, which has been used in recent mesh interconnect designs to resolve contention [30, 10, 11, 40]. 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

[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 mesh-based baselines.

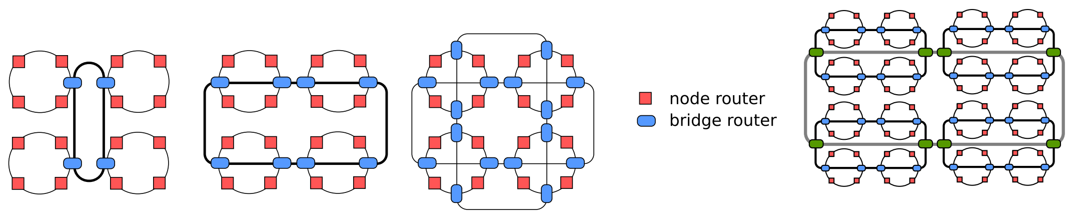
deflections may cause flits to arrive out-of-order (this is fundamental to any non-minimal adaptively-routed network). Because we use Retransmit-Once [10], packet reassembly works despite out-of-order arrival.

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. In our proposed network topology (detailed below in §3.3), we use wider global rings than local rings (analogous to a *fat tree*) 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.)

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 Topology

We now provide an overview of the assumed hierarchical ring topology. We combine multiple *local* rings, each of which connects a subset of the nodes in a system, and connect them with a single *global* ring. Local and global rings connect via *bridge routers*, which attach to two rings (one global and one local). This two-level structure was shown in Fig. 1.



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

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

Figure 6: Hierarchical ring topology of HiRD

Bisection Bandwidth: Note that the fundamental bisection bandwidth of a hierarchical-ring network is still constrained by the single global ring. However, the bandwidth of the global ring can be increased proportional to demand. This is less costly than building a high-bandwidth ring that reaches all nodes in the system, because only the bridge routers require the wider and/or faster datapaths. We adopt a higher-bandwidth global ring to improve network performance in our evaluations (as we specify in §6, we maintain equivalent bisection bandwidth to a mesh for 4x4 and 8x8 design points).

Bridge Router and Global Ring Configurations: When building a two-level topology, 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 tradeoff. In this work, we study three particular configurations, shown in Fig. 6a. 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 §7.5, this choice of topology has a significant impact on system performance. It will also impact network power and total router area. Outside of that sensitivity study, we assume an 8-bridge design for the remainder of this paper.

Generalized Hierarchical Rings: Multiple Levels: Finally, 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 Fig. 6b, to build a 64-node network.

Alternate Topologies: Note that we do not exhaustively study hierarchical topologies that make use of rings, due to space limitations. For example, a hierarchy could be constructed with a global ring at the top level, and mesh networks in local "neighborhoods" joined by this ring. Alternately, local rings could be joined by a global interconnect with some other topology, such as a mesh. Other work has studied how to apply hierarchy in various ways to yield better interconnect designs, e.g. [41, 8]. We leave exploration of alternative hierarchical designs for future work, instead focusing on a new hierarchical ring design that aims to keep all routers in the hierarchy as simple as possible.

3.4 Routing

Finally, we briefly address routing. Because a hierarchical topology 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, it takes any bridge router which connects to a ring with a more specific prefix match, 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.

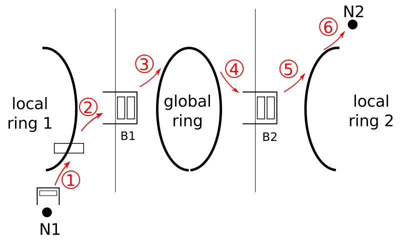


Figure 7: The need for the injection and transfer guarantees: contention experienced by a flit during its journey.

To understand the need for each guarantee, let us consider a simple example, shown in Fig. 7. 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.

We now describe both the injection guarantee (§4.1) and transfer guarantee (§4.2) mechanisms. We qualitatively argue correctness here, in §4.3, and quantitatively evaluate both mechanisms in §7.3.

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 can be intuitively understood to be 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).

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 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 slot 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 a slot must open eventually, and the stuck flit will then take the slot 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 argue the end-to-end delivery guarantee for any flit in the network given the injection and transfer guarantees described above.

Before we argue 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 because of Retransmit-Once [10], as described previously). 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 slot in that transfer queue (except when the slot 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 argument is structured as follows: we first argue that if no new flits enter the network, then the network will drain in finite time. This will then be used to argue that 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, we argue that 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 argue that no livelock will occur.

Any node can inject: Now that we have argued 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 argued), 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 argued 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 slot 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 open slot in the transfer queue. But the transfer guarantee does exactly this by observing ring slots in sequence and reserving transfer queue slots when a flit becomes stuck in a ring. Because the mechanism will eventually observe every slot, 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 buffer in the router. 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 global throttling arbitration logic with only two wires, one to signal blocked injection and one control line that throttles the router. Since 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 Qualitative Comparison to Previous Work

In this section, we qualitatively compare HiRD to four baseline designs: a buffered mesh, a bufferless mesh, a single ring, and a buffered hierarchical ring in terms of (i) system performance and (ii) router and network area and power. In §7, we will perform quantitative evaluations of HiRD against all of these baselines in order to demonstrate the tradeoffs that we discuss in this section.

Buffered Mesh: A mesh interconnect maps naturally to a two-dimensional tiled CMP design. Meshes are thus a common choice for manycore CMPs, such as the Tilera Tile100 [39] and Intel Terascale [19]. A mesh topology has good performance scalability relative to ring-based designs, because its bisection bandwidth grows naturally with the network. In contrast, single- and hierarchical-ring topologies must widen a ring to maintain adequate bisection bandwidth. However, a mesh topology forces cross-chip traffic to visit every node along a path. In contrast, a hierarchical topology such as HiRD allows traffic to cross the chip with fewer hops by using the global ring.

A buffered mesh router has a high complexity and cost compared to the node router at each CPU/cache node in the HiRD NoC. In a buffered mesh, each router contains buffers at every router input, and must also include a crossbar to move flits from any input to any output. In HiRD, a node router only attaches to the ring nodes on either side of the current node, and has no in-ring buffers. Bridge routers have buffering, but the required buffer space is less than in a buffered mesh, and there are fewer bridge routers than nodes in total.

Bufferless Mesh: Bufferless mesh networks (such as CHIPPER [10]) eliminate the buffers from conventional mesh routers, and yield significant energy efficiency improvements at low-to-medium load. The same topology tradeoffs as with the buffered mesh (described above) also apply in this case; the modifications made by a bufferless mesh NoC apply solely to the router design itself.

A bufferless mesh router is conceptually similar to a bridge router in HiRD in that both use deflection, because neither router can exert backpressure and stall traffic on a router input. In the case of HiRD, rings have no in-ring buffers, and so traffic on rings continues to flow. In the case of a bufferless mesh with deflection routing, no router contains in-network buffers, so all traffic moves continuously. In contrast to a bufferless mesh router, a HiRD bridge router contains buffers for traffic that transfers between rings. Both routers contain a crossbar (in the HiRD case, to allow transfer from either local ring direction to any global ring slice). The routing logic in a bufferless mesh router is more complex, because it must consider flit priorities and 2D-mesh routing (see [10, 30] for example routing algorithms). However, despite these tradeoffs, a HiRD network contains fewer bridge routers than nodes, whereas a bufferless mesh contains such a router at every node.

Single Ring: A single ring network has a simpler topology than a hierarchical ring. At low load, and in small networks, a single ring performs well because each hop has a low latency (due to simple routers). We show this quantitatively in §2. However, as we also show, a ring scales poorly relative to other topologies as load increases or as network size increases. At high load, the low bisection bandwidth of a ring becomes the main bottleneck, and ring utilization increases rapidly. As the network scales up, bisection bandwidth is still a bottleneck, but the average number of hops also grows more quickly than in other topologies: traffic that crosses a single-ring network visits half of the nodes during its journey, and this can lead to high latencies and high dynamic link power. Building a network with hierarchy allows express transit on a global ring, and maintains lower ring utilization by segmenting the network into multiple local rings.

The main advantage of a single-ring network is its very simple router. In a ring that has no buffers for traffic in flight, a router can consist simply of several pipeline registers, a MUX to allow traffic injection, and a MUX to allow traffic ejection. Control logic is also simple, because a router only needs to recognize when traffic addressed to the local node arrives on the ring, and does not need to compute a routing function. Control in HiRD is more complex, both because routing between rings involves a routing function, and because ring transfer buffering requires flow control (deflection). However, in exchange for this additional complexity, dynamic power can be lower because network utilization is reduced by using multiple rings and employing a global ring for cross-chip transit.

Buffered Hierarchical Rings: The link topology of HiRD is similar to the state-of-the-art buffered hierarchical ring design against which we compare [33]. The key difference lies in how the routers (both bridges between rings, and node routers around each local ring) are designed. A buffered hierarchical ring network places local buffering at each

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
Cache coherence	directory-based protocol (based on SGI Origin [28, 5]), directory entries colocated with shared cache blocks
Simulation length	5M cycle warm-up, 25M cycle active execution

Table 1: Simulation and system configuration parameters.

Parameter	Network	Setting
Interconnect Links	Buffered Mesh	1-cycle latency, 64-bit width
	Bufferless Mesh	
	Single Ring	Bidirectional, **4x4:** 128-bit width, **8x8:** 256-bit width
	Buffered HRing	Bidirectional, **4x4:** two-level hierarchy, 128-bit local and global rings, **8x8:** three-level hierarchy, 256-bit local and global 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	Buffered Mesh	3-cycle per-hop latency, 8 VCs, 8 flits/VC, buffer bypassing [43], X-Y routing, age-based arbitration
	Bufferless Mesh	CHIPPER [10], 3-cycle per-hop latency
	Single Ring	1-cycle per-hop latency (as in [25])
	Buffered HRing	Node (NIC) and bridge (IRI) routers based on [33]; 4-flit in-ring and transfer FIFOs. Bidirectional links of dual-flit width (for fair comparison with our design). Bubble flow control [3] for deadlock freedom.
	HiRD	local-to-global buffer depth 1, global-to-local buffer depth 4

Table 2: Network parameters.

node router in a ring. In-ring buffers allow the use of flow control, and bridge routers make use of this flow control to stall incoming traffic on a ring when a transfer buffer becomes full. In contrast, HiRD has no flow control in any ring, and must make use of deflections when a transfer buffer fills. Placing buffers in the ring avoids deflections (which reduce performance and increase link power) but adds significant buffer power and area to every node router, and also requires flow-control logic at every router in the ring.

6 Methodology

We perform our evaluations using a cycle-accurate simulator of a CMP with interconnect to provide application-level performance results. Details are given in Tables 1 and 2. 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 [28]) is modeled on top of the cycle-accurate interconnect model. Each core models an out-of-order instruction window, and models its MSHRs (Miss-Status Handling Registers [27], which track outstanding cache-miss requests) realistically, including the additional request-buffering overhead and possible stalls due to a congested network. 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, and is also used in e.g. [30, 10]. Instruction traces for the simulator are taken using a Pintool [29] on representative portions of SPEC CPU2006 workloads [37].

We compare to several baselines: a conventional virtual-channel buffered mesh, a bufferless mesh (CHIPPER [10]), a a single bidirectional ring, a state-of-the-art buffered hierarchical ring [33], a concentrated mesh, and a torus topology [6]. To fairly compare networks, we hold bisection bandwidth constant. This comparison is fair because the modeled hardware overhead (power, die area, 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 [6], and uses buffer bypassing [43]. Our goal is to compare against a representative network of each topology and

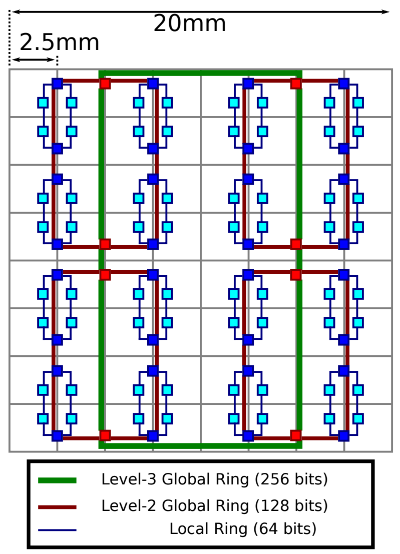

20mm
2.5mm
Level-3 Global Ring (256 bits)
Level-2 Global Ring (128 bits)
Local Ring (64 bits)

Figure 8: Assumed floorplan for HiRD 3-level (64-node) network. Two-level (16-node) network consists of one quadrant of this floorplan.

design philosophy.

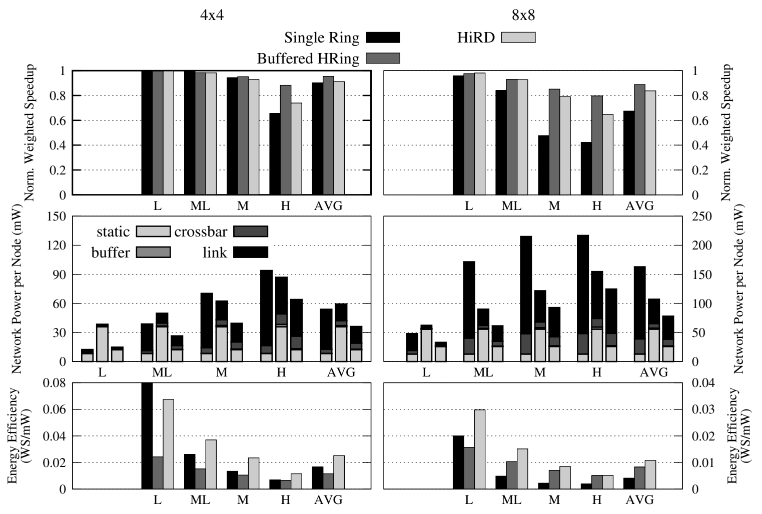
Application & Synthetic Workloads: We evaluate each system design with a set of 60 multiprogrammed workloads. Each workload consists of one single-threaded instance of a SPEC CPU2006 [37] benchmark on each core, for a total of either 16 (4x4) or 64 (8x8) benchmark instances per workload. Multiprogrammed 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).

To show that our conclusions are robust against particular application 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 and/or other datapath components, buffers, and control logic. For links, buffers and datapath elements, we use ORION 2.0 [42]. Control logic is modeled in Verilog synthesized with a commercial 65nm design library.

We assume a 2.5 mm link length for mesh and single ring topologies. For the hierarchical ring design, we assume 1 mm links between local-ring routers, since 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, since 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 Fig. 8.

Application Evaluation Metrics: We present application performance results using the commonly-used Weighted Speedup metric [36] (representing system throughput [9]), which calculates performance of each application in a multi-programmed mix relative to how well the application performs when running alone on the system: $WS = \sum_{i=1}^{N} \frac{IPC_i^{shared}}{IPC_i^{alone}}$. We take an application's performance running on a baseline CHIPPER network as a common baseline when computing weighted speedup on all topologies, in order to allow for fair comparison across topologies.


(a) HiRD as compared to buffered hierarchical rings and a single-ring network.

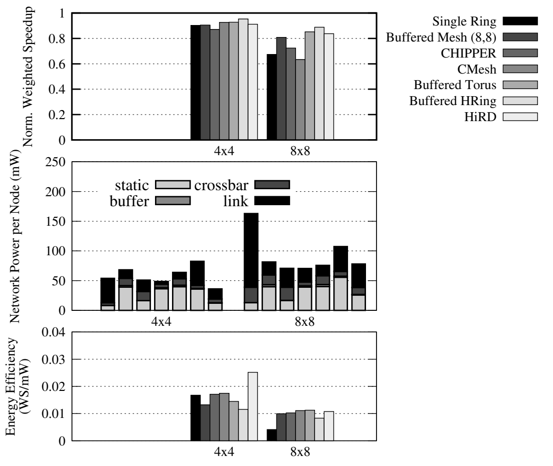

(b) Evaluation against a variety of baseline designs.

Figure 9: Evaluations of application performance, total network power, and energy efficiency.

Configuration	Maximum Transfer FIFO Wait (cycles)	Maximum 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	0.16	0.00	0.16
With Guarantees	1.2 / 66	2.8 / 18	0.13	0.08	0.12

Table 3: Results of worst-case traffic pattern without and with injection/transfer guarantees enabled.

7 Evaluation

7.1 Ring-based Network Designs

Fig. 9a 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, using identical topologies, as well as a single ring (with equalized bisection bandwidth). This comparison is meant to illustrate tradeoffs in *ring-based* networks (comparisons to other topologies will come in §7.2). 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. In the data shown, the buffered hierarchical ring improves performance by 35% (and HiRD by 12.8%) over the single ring in high-load workloads at 16 nodes. The hierarchical topology also reduces power, because hop count is reduced and so link power reduces significantly with respect to a single ring.

2. HiRD maintains a significant fraction of the performance of the buffered hierarchical ring (and hence, advantage over a single ring) in larger networks, where the scaling advantage of hierarchy becomes more important. On average in the 8x8 configuration, the buffered hierarchical ring network obtains 31.6% better application performance than the single ring, while HiRD attains 24.0%.

3. Compared to the buffered hierarchical ring, HiRD has significantly lower network power. On average, HiRD reduces total network power (links and routers) by 39.0% (4x4) or 27.2% (8x8) relative to this baseline. This reduction in turn yields significantly better energy efficiency. Overall, HiRD is the most energy-efficient of the ring-based designs evaluated in this paper in both 4x4 and 8x8 network sizes.

7.2 Alternative Topologies: Mesh, Ring, and Others

Fig. 9b shows a comparison of HiRD against a single ring, a mesh with VC-buffered routers [6], a bufferless mesh (CHIPPER [10]), a concentrated mesh, and the buffered hierarchical ring network [33].[3] As before, we show weighted speedup (application performance) and total network power, both normalized per node, as well as energy efficiency (performance per watt). Several main conclusions can be drawn:

1. Overall, HiRD is competitive in performance with all baselines. In particular, it has 3.6% (0.8%) better performance than the buffered mesh in 4x4 (8x8) networks. Thought it is outperformed slightly by (i) a buffered torus (both network sizes) and (ii) a concentrated mesh (in 4x4), both of these baseline designs require more complex mesh routers.
2. HiRD reduces network power significantly relative to the 4x4 mesh and torus routers, and is nearly on-par with these router designs in 8x8 networks. In the larger configuration, relative to the mesh routers, HiRD reduces static power (because mesh router crossbars are eliminated) but slightly increases link power (both because of ring deflections and because the path between two nodes may not be as direct as in a mesh).
3. Overall, HiRD is the most energy-efficient network design in 4x4 networks (by 44.7% over the most energy-efficient baseline, the concentrated mesh), and is within 4.6% of the buffered torus (the most energy-efficient baseline) in 8x8.

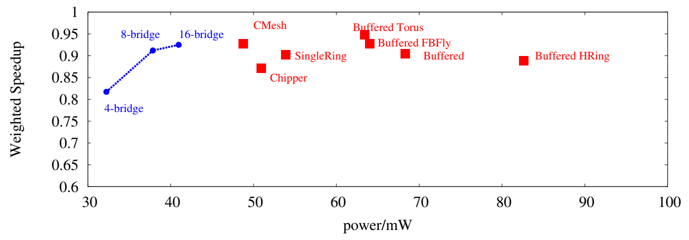


Figure 10: Weighted speedup (Y) vs. power (X) for 4x4 networks.

Fig. 10 highlights the energy-efficiency comparison by showing weighted speedup (Y axis) against power (X axis) for all evaluated networks in 4x4. HiRD is shown at three data points with the three bridge-router configurations (described in §3.3). The top-left is the ideal corner (high performance, low power). As the results show, all three configurations of HiRD are more energy efficient than the evaluated baseline designs at this network size.

Overall, we conclude that HiRD either outperforms or is on-par with other baseline routers with respect to performance and energy efficiency. In cases where its efficiency is close to other designs, the other designs require more complex mesh routers, and HiRD's design simplicity remains an advantage, because it reduces both design/validation effort and router die area.

7.2.1 Synthetic-Traffic Network Behavior

Fig. 11 shows 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 only uniform random traffic for space reasons (other patterns yield similar conclusions overall). Sweeps terminate at network saturation. HiRD performs very closely to the buffered mesh design in 4x4, saturating at nearly the same injection rate but maintaining a slightly lower network latency at high load. The buffered hierarchical ring saturates at a similar point to HiRD and the buffered mesh, 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, closer to the bufferless mesh design, CHIPPER. As network size scales to 8x8, HiRD performs significantly better relative to the meshes, because the hierarchy reduces the cross-chip latency while preserving bisection bandwidth.

7.3 Injection and Transfer Guarantees

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

[3]We also compare to a flattened butterfly [24] VC-buffered configuration, but only for 4x4 (as the routers become very large at 8x8): the FBFly topology yields 4.7% better performance than the buffered mesh on our workloads at 7.1% less power. However, HiRD's substantial power and energy-efficiency advantage remains (68% better energy efficiency). For uniformity, we omit FBFly from our main results plots.

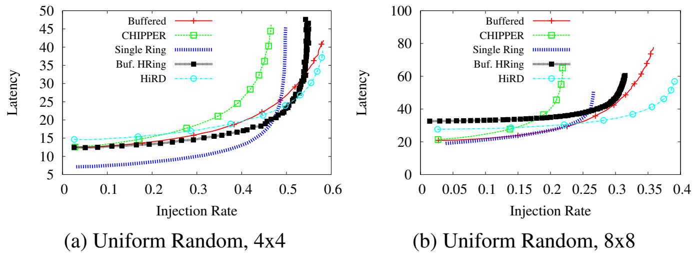


Figure 11: Synthetic-traffic evaluations for 4x4 and 8x8 networks.

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 interjects).
Results: Table 3 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 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.

7.4 Router Area and Timing

We show both critical path length and normalized die area for buffered and bufferless mesh, single-ring, buffered hierarchical ring, and HiRD, in Table 4. Area results are normalized by the buffered mesh baseline, and are reported for all routers required by a 16-node network (e.g., for HiRD, 16 node routers and 8 bridge routers). Note that we do not report a critical path length for the baseline buffered hierarchical ring router, because we do not model its control logic but only its crossbar, buffers, and datapath. This is a conservative assumption which benefits this prior work's area and power results relative to our own, because its control logic also must handle flow control, unlike HiRD's. The buffered hierarchical ring router's critical path will be longer for this reason (because it must check whether credits are available for a downstream buffer).

Two observations are in order. First, hierarchical ring designs in general significantly reduce area relative to the mesh routers, because the node router required at each network node is much simpler and does not require a crossbar or complex routing logic. HiRD reduces total router area by 74.7% from the buffered mesh. Its area is higher than the bufferless mesh router baseline 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 critical path is shorter in HiRD than in mesh-based designs because ring routers have simpler routing and arbitration logic. 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).

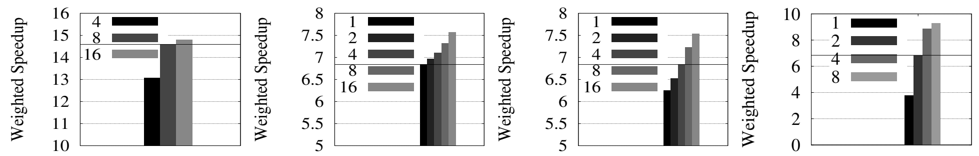
Metric	Buffered	CHIPPER	Single-Ring	Buffered HRing	HiRD
Critical path length (ns)	1.21	0.93	0.33	–	0.61
Normalized area	1	0.232	0.233	0.373	0.353

Table 4: Total router area (16-node network) and critical path.

7.5 Sensitivity Studies

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 Fig. 6a, 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 (Fig. 6b). In Fig. 12a, 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.

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. 12b and 12c respectively, for the 8-bridge 4x4-node topology. 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 result in transfer deflection rates of 28.2% (global-to-local) and 34% (local-to-global).



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

Figure 12: Performance sensitivity to buffer sizes and the global ring bandwidth in a 4x4 network.

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 [34]. In Fig. 12d, 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 §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.

8 Related Work

To our knowledge, HiRD is the first on-chip network design which combines a hierarchical ring topology with simple, deflection-based ring transfers.

Hierarchical Interconnects: Hierarchical ring-based interconnect was proposed in a previous line of work [33, 45, 17, 34, 14]. 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 [6] and bufferless deflection mesh routers [30, 10], and our design leads to significantly simpler router design with reduced area and network power, as we show in this paper.

Udipi et al. proposed a hierarchical topology using global and local buses [41]. While this work recognizes the benefits of hierarchy, using buses limits scalability in favor of simplicity. In contrast, our design has more favorable scaling, in exchange for using flit-switching routers. Das et al. examined several hierarchical designs, including a concentrated mesh (one mesh router shared by several nearby nodes) [8]. We compared to a concentrated mesh in §7.2.

One previous system, SCI (Scalable Coherent Interface) [15], 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) 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.

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 [30, 10]. As we show in our evaluations for CHIPPER [10], while such designs successfully reduce power and area, they retain the fundamental complexity of mesh-based routers, and hierarchical ring designs are a better overall performance/power tradeoff for most workloads.

Other Ring-based Topologies: Spidergon [4] 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 [22] 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: Finally, Kim [23] 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, because a turn is possible at any node in a mesh, every router requires such a buffer; in contrast, we require these transfer buffers only at bridge routers, and their cost is amortized over all nodes. Mullins et al. [31] propose a buffered mesh router with single-cycle arbitration. We share the goal of building a simpler, faster router. Our work differs in that rather than simplifying a router within an existing topology, we propose a novel combination of topology, flow control/routing, and router design that yields a low-cost, high-performance network.

9 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 topology yields good performance and scalability relative to both a single ring and a mesh. HiRD has two new contributions: (1) a simple router design which 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* which ensure deterministic 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 with less complexity than other alternatives.

References

[1] P. Baran. On distributed communications networks. *IEEE Trans. on Comm.*, 1964. 2, 6

[2] S. Borkar. NoCs: What's the point? NSF Workshop on Emerging Tech. for Interconnects (WETI), Feb. 2012. 2

[3] C. Carrión et al. A flow control mechanism to avoid message deadlock in k-ary n-cube networks. *High Performance Computing*, 1997. 12

[4] M. Coppola et al. Spidergon: a novel on-chip communication network. *Proc. Int'l Symposium on System on Chip*, Nov 2004. 18

[5] D. E. Culler et al. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999. 12

[6] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004. 2, 4, 12, 14, 18

[7] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *DAC-38*, 2001. 1

[8] R. Das et al. Design and evaluation of hierarchical on-chip network topologies for next generation CMPs. *HPCA-15*, 2009. 4, 8, 18

[9] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28:42–53, May 2008. 13

[10] C. Fallin et al. CHIPPER: A low-complexity bufferless deflection router. *HPCA-17*, 2011. 2, 6, 7, 10, 11, 12, 14, 18

[11] C. Fallin et al. MinBD: Minimally-buffered deflection routing for energy-efficient interconnect. *NOCS 2012*, 2012. 6

[12] C. Gómez et al. Reducing packet dropping in a bufferless noc. *Euro-Par-14*, 2008. 2

[13] P. Gratz, C. Kim, R. McDonald, and S. Keckler. Implementation and evaluation of on-chip network architectures. *ICCD*, 2006. 2

[14] R. Grindley et al. The NUMAchine multiprocessor. *In Proc. 29th Int'l Conf. on Parallel Processing*, pages 487–496, 2000. 2, 4, 18

[15] D. Gustavson. The scalable coherent interface and related standards projects. *IEEE Micro*, 12:10 – 22, Feb. 1992. 18

[16] K. Hale et al. Segment gating for static energy reduction in networks-on-chip. *NoCArc*, 2009. 2

[17] V. C. Harmacher and H. Jiang. Hierarchical ring network configuration and performance modeling. *IEEE Transaction on Computers*, 50(1):1–12, 2001. 2, 4, 18

[18] M. Hayenga, N. E. Jerger, and M. Lipasti. Scarab: A single cycle adaptive routing and bufferless network. *MICRO-42*, 2009. 2

[19] Y. Hoskote et al. A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro*, 2007. 2, 11

[20] 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`. 2

[21] S. Jafri et al. Adaptive flow control for robust performance and energy. *MICRO-43*, 2010. 2

[22] F. Karim et al. On-chip communication architecture for oc-768 network processors. *DAC-38*, 2001. 18

[23] J. Kim. Low-cost router microarchitecture for on-chip networks. *MICRO-42*, 2009. 2, 6, 18

[24] J. Kim and W. Dally. Flattened butterfly: A cost-efficient topology for high-radix networks. *ISCA-34*, 2007. 14

[25] J. Kim and H. Kim. Router microarchitecture and scalability of ring topology in on-chip networks. *NoCArc*, 2009. 2, 12

[26] S. Konstantinidou and L. Snyder. Chaos router: architecture and performance. *ISCA-18*, 1991. 2

[27] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. *ISCA-8*, 1981. 6, 12

[28] J. Laudon and D. Lenoski. The SGI Origin: a ccNUMA highly scalable server. *ISCA-24*, 1997. 12

[29] C.-K. Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. *PLDI*, 2005. 12

[30] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. *ISCA-36*, 2009. 2, 6, 11, 12, 18

[31] R. Mullins et al. Low-latency virtual-channel routers for on-chip networks. *ISCA-31*, 2004. 18

[32] D. Pham et al. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *J. Solid-State Circuits*, 41(1):179–196, Jan 2006. 2

[33] G. Ravindran and M. Stumm. A performance comparison of hierarchical ring- and mesh-connected multiprocessor networks. *HPCA-3*, 1997. 2, 3, 4, 5, 11, 12, 14, 18

[34] G. Ravindran and M. Stumm. On topology and bisection bandwidth for hierarchical-ring networks for shared memory multiprocessors. *HPCA-4*, 1998. 2, 4, 17, 18

[35] L. Seiler et al. Larrabee: a many-core x86 architecture for visual computing. *SIGGRAPH*, 2008. 2

[36] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. *ASPLOS-9*, 2000. 13

[37] Standard Performance Evaluation Corporation. SPEC CPU2006. `http://www.spec.org/cpu2006`. 12, 13

[38] M. Taylor, J. Kim, J. Miller, and D. Wentzlaff. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, Mar 2002. 2

[39] Tilera Corporation. Tilera announces the world's first 100-core processor with the new TILE-Gx family. `http://www.tilera.com/news_&_events/press_release_091026.php`. 2, 11

[40] S. Tota et al. Implementation analysis of NoC: a MPSoC trace-driven approach. *GLSVLSI-16*, 2006. 2, 6

[41] A. Udipi et al. Towards scalable, energy-efficient, bus-based on-chip networks. *HPCA-16*, 2010. 4, 8, 18

[42] H. Wang et al. Orion: a power-performance simulator for interconnection networks. *MICRO-35*, 2002. 13

[43] H. Wang, L. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. *MICRO-36*, 2003. 12

[44] D. Wentzlaff et al. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007. 2

[45] X. Zhang and Y. Yan. Comparative modeling and evaluation of CC-NUMA and COMA on hierarchical ring architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 6(12):1316 –1331, Dec 1995. 2, 4, 18