High-Radix On-chip Networks with Low-Radix Routers

Animesh Jain, Ritesh Parikh and Valeria Bertacco
Department of Computer Science and Engineering, University of Michigan
{anijain, parikh, valeria}@umich.edu

***Abstract*—Networks-on-chip (NoCs) have become increasingly widespread in recent years due to the extensive integration of many components in modern multicore processors and SoC designs. One of the fundamental tradeoffs in NoC design is the radix of its constituent routers. While high-radix routers enable a richly connected and low diameter network, low-radix routers allow for a small silicon area. Since the NoC consumes a significant portion of the on-chip resources, naïvely deploying an expensive high-radix network is not a practical option.**

In this work, we present a novel solution to provide high-radix like performance at a cost similar to that of a low-radix network. Our solution leverages the irregularity in runtime communication patterns to provide short low-latency paths between frequently communicating nodes, while infrequently communicating pairs rely on longer paths. To this end, it leverages a flexible topology reconfiguration infrastructure with abundantly available links between routers (in accordance to a high-radix topology) that are decoupled from scarcely available router ports (similar to a low-radix topology). Network links are bound to router ports at runtime to form connected and deadlock-free topologies. Binding selections are based on the traffic patterns observed, which are synthesized through a distributed statistics-collection framework. Our experiments on a 64-node CMP, running multiprogrammed workloads, show that we can reduce average network latency by 19% over an area- and power- comparable mesh NoC. The latency improvements for non-uniform synthetic traffic are above 30%.

I. Introduction

As a result of increasing integration of components into CMP and SoC architectures, networks-on-chip (NoCs) have become the dominant choice for on-chip interconnects, due to the highly concurrent communication paths and better scalability they provide. Moreover, to keep up with the communication demands of the cores/IPs on-chip, NoCs are increasingly incorporating bulky and power-hungry resources, required to meet target latency and bandwidth goals.

A key design choice in this context is the radix of the network routers, that is, the number of I/O ports that a router provides to connect to adjacent routers High-radix routers (>5 I/O ports) enable low-diameter topologies, allowing all processing nodes to be reached in just a few hops from any source. However, router components, such as the crossbar and the allocators, grow in area quadratically with the radix of the router. In addition, high-radix routers lead to increased signal propagation latencies, and slower operating frequencies. A popular alternative are topologies deploying low-radix routers (<5 I/O ports), such as meshes: they can operate at higher frequencies and use less area and power. For example, a radix-7 router requires a 4.1% greater cycle time than a radix-5 router. Unfortunately, low-radix topologies could lead to large network diameter and prohibitively high hop counts. They are especially problematic for applications that do not have sufficient memory-level parallelism (MLP) to hide their higher latency.

To overcome the limitations above, in this work we present HiROIC (**Hi**gh **R**adix **O**n-chip Networks at **I**ncremental re-**C**onfiguration Cost). With HiROIC we want to provide the best of both classes: the effective network diameter of high-radix topologies and the low resource requirements of low-radix networks. HiROIC exploits the non-uniformity of communication patterns to provide short, low latency paths only between heavily communicating nodes, while it forces low volume source-destination pairs to use longer paths. With the increasing integration of application-specific components [4], the location and quantity of heavily used routing paths is likely to be highly unbalanced both across and within applications. In such designs, only a small subset of accelerators will be active at any point in time, and this subset will actively communicate with memories banks distributed across the chip. Being able to optimize the latency between selected nodes (active accelerators and memory banks) at runtime, will greatly benefit such designs. We therefore envision great potential for the deployment of HiROIC in upcoming CMP and SoC designs.

HiROIC leverages routing and topology reconfigurations to optimize connectivity for high-volume source-destination pairs. At the heart of HiROIC is the concept of port-link decoupling: network links are connected to routers' ports only at runtime, and the binding is modified dynamically based on the changes in traffic patterns imposed by the application. Our NoC design includes low-radix routers, but abundant links, as in high-radix topology, so to *potentially* provide short paths between any source-destination pair. In HiROIC, computation is partitioned into epochs of execution, with port-link binding fixed during each epoch. At the end of an epoch, the mapping is re-evaluated based on the observed traffic patterns, and modified if there is space for improvements. While HiROIC's wiring overhead is greater than conventional topologies (*e.g.*, meshes), we observe that wires do not constitute a timing bottleneck in conventional router pipelines [10]. Note that, in typical NoCs, routers have one *local* port connecting to the processing node(s). Since this connection is essential, HiROIC uses a fixed port-link binding for *local* ports. In the rest of this paper we exclude the local port(s) when reporting the radix of the router.

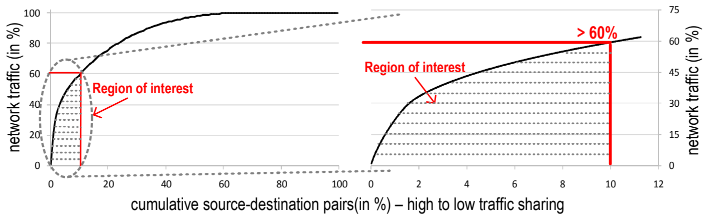

Fig. 1: **Network activity shared by the most exercised source-destination pairs.** The plot on the right is an enlargement of the one on the left. The top 10% source-destination pairs are collectively responsible for more than 60% of the total network traffic.

It is essential for HiROIC to have a high variation between high-usage source-destination pairs and other source-destination pairs. To this end, we conducted a study whose findings are plotted in Figure 1. The plot shows the contribution of traffic flowing between each source-destination pair. Our testbed consisted of an 8x8 mesh CMP running a multiprogrammed mix of applications from the SPEC CPU2006 suite. Source-destination pairs are sorted by decreasing traffic activity during the execution, and the plot on the left indicates what fraction of network traffic (Y axis) was carried out by a given fraction of sorted pairs. The plot on the right is an enlargement of the contribution by the top 12% source-destination pairs: less than 10% of them share as much as 60% of the traffic load on average. Beyond the tenth percentile of utilization, this disparity is no longer obvious. Thus, HiROIC's goal is to leverage the 10% most used pairs to provide short and high-bandwidth paths between them. This, in turn, minimizes the effective network hop count.

Contributions. In summary, the novel contributions of this work are:

- A **router architecture** to mimic the high-radix routers' connectivity while consuming resources comparable to a low-radix router.
- A distributed, deadlock-free **reconfiguration algorithm** to predict an application's future communication needs and optimize the network topology to provide short paths between high-traffic source-destination pairs.

In our evaluation with non-uniform multiprogrammed workloads from the SPEC CPU 2006 suite, HiROIC's 64-node layout reduces average network latency by 19% compared to a baseline mesh NoC. For non-uniform synthetic traffic, latency improves from 30% to 38%, depending on physical topology and traffic injection rate.

II. Related Work

Much of the research targeting performance improvements in NoC designs has focused on: i) reducing the number of pipeline stages within the router [14], and ii) increasing the clock frequency of the router's operation [2]. Other works strove to optimize the router design for specific topologies and flow control [9]. Our work leverages an orthogonal approach to improve performance – decreasing the average packet's hop count. Previous works that leveraged application-driven configuration for the NoC targeted the design phase of the NoC, with no ability to reconfigure at runtime. These solutions would characterize all applications that were expected to run on the system and then, based on the analysis, optimize the design's: i) topology [17], ii) routing [12], *etc.* In contrast, HiROIC adapts dynamically to changing application patterns and reconfigures the topology at runtime.

Runtime reconfiguration solutions have also been proposed to optimize either power or performance. To reduce NoC leakage power,

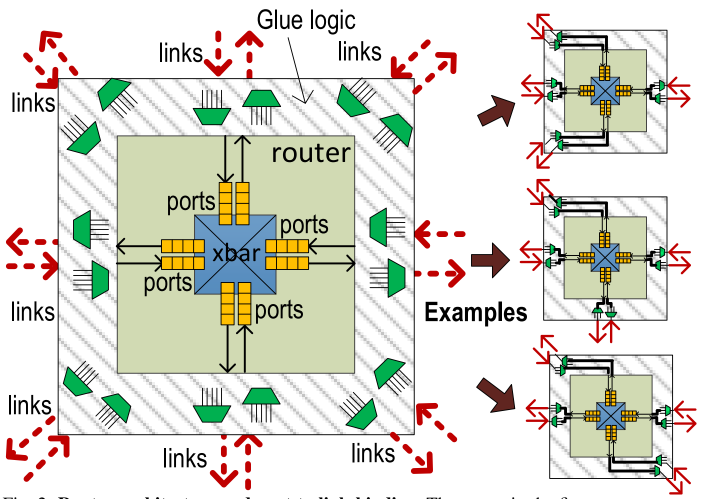

Fig. 2: **Router architecture and port-to-link binding.** The router in the figure can connect its four router ports to eight available neighboring links, by assigning the multiplexers' select signals accordingly. On the right we show 3 such possible bindings.

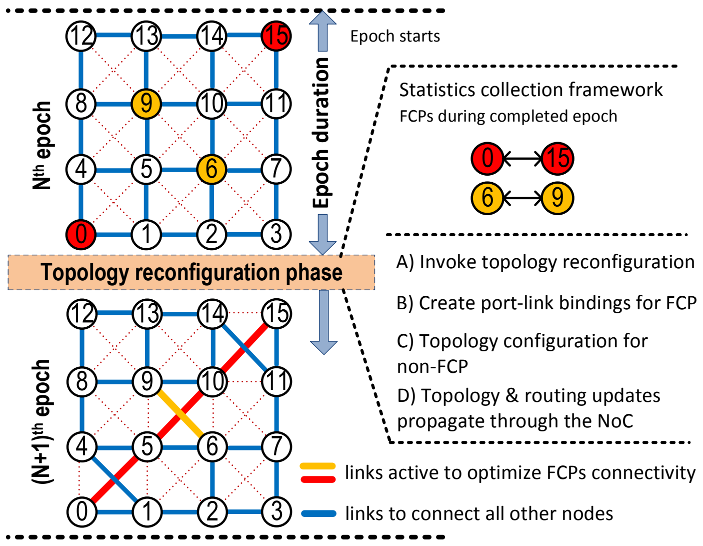
Epoch starts
Nth epoch
Epoch duration
Statistics collection framework
FCPs during completed epoch
Topology reconfiguration phase
A) Invoke topology reconfiguration
B) Create port-link bindings for FCP
C) Topology configuration for non-FCP
D) Topology & routing updates propagate through the NoC
(N+1)th epoch
links active to optimize FCPs connectivity
links to connect all other nodes

Fig. 3: **HiROIC execution flow.** Application's execution is partitioned into epochs. During each epoch, HiROIC monitors the NoC's traffic patterns. At the end of the epoch, the data is used to determine whether a topology reconfiguration is appropriate. If so, the new configuration aims at minimizing the distance between FCPs.

researchers have proposed power-gating network resources at a coarse or fine granularity, during periods of inactivity. Other examples include [8], which reconfigures routing at runtime to provide better management of hotspots, and [7], which improves performance by adapting the channel's bandwidth. All these techniques provide valuable benefits and can be deployed concurrently with HiROIC, which targets dynamic topology reconfiguration, to attain additional gains. There is also a body of work related to runtime topology reconfiguration whose goal, however, has been reliability. In these solutions, for instance Ariadne [1] and uDIREC [13], the topology changes because of a runtime fault, and the authors propose reconfiguration techniques to update the routing function. HiROIC's key contribution is a dynamic topology reconfiguration solution; upon each reconfiguration we also must perform a routing reconfiguration, based on the solution in [1].

III. Methodology

Our solution leverages port-link decoupling to mimic high-radix topologies, while utilizing an amount of resources comparable to a low-radix topology. To optimize the topology for high-volume communication patterns, we perform the following steps: i) we collect traffic statistics over execution intervals (epochs) to predict future traffic behavior, ii) we trigger topology reconfigurations when we observe pattern changes, and iii) we set port-link bindings at each router based on the new topology planned.

In conventional networks, router ports have fixed one-to-one mapping with links. In contrast, we propose to provide more links than those available in low-radix topologies, eliminating the traditional fixed connections. At runtime, router ports are bound to a subset of the available links, based on the application's communication demands. The internal micro-architecture of the router is not modified, with the exception of the necessary updates to the routing tables, based on the selected configuration. Figure 2 shows on the left the schematic of a four-port HiROIC-enabled router with the opportunity to bind to eight links. The *glue logic* in the hashed area comprises multiplexers to complete the bindings. The right side of the figure presents three examples of port-to-link bindings for the router.

HiROIC's Execution Flow. In HiROIC, the NoC's execution is partitioned into epochs. During each epoch, our distributed traffic-statistics collection framework monitors the density of communication between all source-destination pairs. Our goal is to identify the pairs that transfer the majority of the traffic (see also Figure 1) so as to minimize their hop count. In the rest of this paper, we will refer to any such pair as the *Frequently Communicating Pair (FCP)*. At the end of each epoch, we analyze the composition of the FCP set and determine whether a topology reconfiguration should occur to improve on the current port-link binding. Figure 3 provides an overview of this process.

Note that our approach strives to predict application's demands based on the traffic observations during the current epoch. This is a valid approximation, as long as our epochs are short compared to the frequency of major phase changes in the application's behavior. We observe that, in practice, applications' phases are at least hundreds of thousands of cycles long ; thus, in our evaluation we set the epoch length to 10,000 cycles, so to be able to quickly respond to significant traffic pattern changes. In the next sections we present a distributed, fast and low-overhead implementation of this collection and reconfiguration process. Note that this reconfiguration process operates mostly in the background, with minimal impact to mainstream network operation and only during transitions between topologies.

IV. Topology Reconfiguration

Considering the complexity of the decision and reconfiguration process, at first a software implementation would seem a prudent choice. However, it would require collecting all traffic statistics at a central processing node, where a dedicated software routine ranks source-destination pairs by traffic density, determines if a topology reconfiguration should occur and what the new setup should be; then the new binding information should be distributed to the NoC routers. The cost in latency and dedicated resources of this approach makes it impractical in the context of the epoch sizes we target.

Thus, we opted to implement HiROIC using lightweight, distributed hardware components implementing an approximation of the ideal algorithmic solution, for the benefits of design simplicity and to attain low-latency reconfigurations. The hardware components added to each router are shown in Figure 4. In addition we leverage a minimalistic *topology-generator network* to control the distributed construction of new topologies. We describe this solution below and provide insights on the hardware additions in the next section.

Port-link bindings for FCPs. During each epoch, we collect traffic statistics at each router in the *traffic directory* unit. This unit counts the packets received from each source, and thus allows us to determine the frequently communicating pairs (FCPs) in a distributed fashion. The selection is made by comparing the number of packets received from each source against a network-wide threshold, called HT_{th}, which varies dynamically. Once the FCP set for each router is determined, the next step is to enable the NoC's links that facilitate low-latency communication for those critical pairs. These links are selected by traversing the network, router by router, and building low-latency paths connecting the FCPs.

The traversal is performed router by router, starting from the router connected to processing node 0, and with each router providing some (or none) of the FCP. At any particular time, the router providing the information about the FCPs to connect is referred to as the *controller router*. For each FCP entry from a controller router, HiROIC attempts to enable all the links between the source and the controller router. This is achieved by using the low-overhead topology-generator network, and a small logic block that activates the appropriate select signals on the multiplexers, so to enable the links on the shortest path between the relevant source and destination routers.

The controller router chooses first a FCP entry and then performs the port-link binding to enable the adjacent link on the shortest path from the source node. The port-link binding activity is then transferred to the neighboring router on the shortest path by using the link just enabled.

In turn, the neighbor router performs the port-link binding along this designated shortest path. This process continues until the source node is reached. The control is then transferred back to the controller router, which moves on to perform port-link bindings for the next FCP entry. Once the controller router has completed all the bindings for its FCP entries, the control moves to the next router. This completes the first phase of the topology reconfiguration algorithm.

Port-link bindings for non-FCPs. In our distributed algorithm, we use aggressive settings for the HT_{th} threshold, thus it is often the case that we can enable all the links required to provide shortest paths for the FCPs, and still have several disconnected ports in a number of routers. Therefore, the second phase in topology reconfiguration binds the free router ports to links that are still available. This is also achieved by traversing the network, router by router. Each router chooses among locally-available port-link bindings in a greedy fashion until it has bound all its ports, or it has run out of binding options. While this greedy approach does not lead to an optimal mapping with maximum port-link bindings, it still provides very good solutions, as our experiments show that 94% of the routers are able to bound all their ports to links.

Note that this kind of greedy approach to binding ports to links may not always result in a fully connected network. We ran several Monte Carlo simulations to evaluate the frequency of this situation and found that disconnected networks are in practice a rare event, occurring only 5 times in 1,000 topology reconfigurations. Fortunately, this situation can be easily detected when we apply the routing reconfiguration algorithm to the new topology: if any destination router is not reachable in at least one routing table, the network is disconnected. When this situation occurs, HiROIC broadcasts a 1-bit *disconnect signal* to all the routers through the topology-generator network. Upon reception of the signal, all routers revert the topology back to a baseline low-radix topology (a 2D mesh in our evaluation). To perform this step quickly, HiROIC maintains a register at each router storing port-link bindings corresponding to the baseline topology. Overall, since this is a rarely occurring situation, it does not significantly affect HiROIC's overall adaptivity to application's communication needs.

Routing in the new topology. After the generation of the new topology is complete, all routing paths must be updated. To this end, we leverage Ariadne's route-reconfiguration algorithm [1]: Ariadne was proposed for reconfiguration around faulty components and, due to the increasing reliability concerns with shrinking transistor sizes, we assume the Ariadne functionality to be already present on-chip. Ariadne leverages the up*/down* algorithm [16] for routing in irregular networks, while proposing a quick and lightweight distributed implementation to update routes upon each topology change. Ariadne is reported to reconfigure a 64-node network in only $\sim$4K cycles, and it is therefore an ideal fit for HiROIC, if reconfiguration is triggered once every few epochs.

A. Topology Configuration Performance

The time required to complete topology reconfigurations is driven by three factors: i) time to build a new topology ($\sim$500 cycles), ii) time to calculate new routes ($\sim$4K cycles), and ii) time to drain packets that were in flight during the reconfiguration – so to avoid deadlocks, lost or dropped packets ($\sim$200 cycles). We propose to hide the latency imposed by the first and second factor by using duplicate routing tables and link-to-port configuration registers. We can use this shadow set of storage to compute topology and routing tables in the background, while communication proceeds in the old topology. We then copy over the new configuration and routes, once their generation is complete.

In addition, to switch topology configurations quickly and still avoid lost and dropped flits, we assume a router design where the size of the input buffers is a multiple of the number of flits possible in a packet. With this assumption, any time we need to switch to a new topology, we advance communication for a few cycles, until each packet sits entirely in one router. This goal can be accomplished by forbidding packets to access new buffers by temporarily disabling the virtual channel allocation (VA) unit. In the worst case scenario, where each packet is waiting for traversal of the packet at its downstream router, it will take 64*3=192 cycles for a 64-node system with 3-stage routers to complete this task. At that point, we can switch to the new configuration.

Reconfiguration-induced deadlocks. HiROIC's reconfiguration algorithm can cause routing deadlocks even if both the old (before topology reconfiguration) and new (after topology reconfiguration) routing functions are independently deadlock-free [11]. Such deadlocks can be easily detected by identifying packets that are requesting illegal turns under the new topology. We eject such packets to the network interface at the local router port, and then re-inject them into the network upon buffer availability. Ariadne [1] utilizes a similar technique to overcome reconfiguration-induced deadlocks.

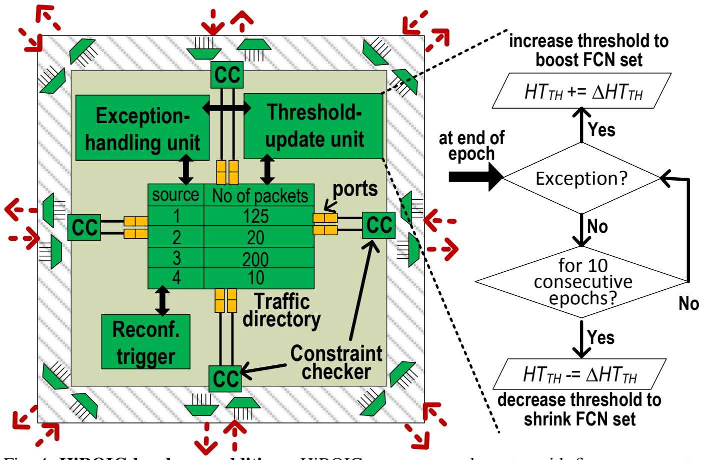

Fig. 4: **HiROIC hardware additions.** HiROIC augments each router with five components (shown in dark green): i) a traffic directory to count packets received from other routers, ii) a reconfiguration trigger unit, iii) constraint checkers to check whether a port is already bound to a link, iv) an exception-handling unit to detect execution anomalies, and v) a threshold-update unit to adjust the packet threshold according to application needs. The right side of the figure shows the threshold-update algorithm.

V. Hardware Additions

As mentioned earlier, our implementation consists only of simple adders, comparators and storage structures, distributed in the NoC. As illustrated in Figure 4, HiROIC's hardware implementation consists of five components at each router: i) a directory to maintain per-destination traffic statistics, ii) a reconfiguration-trigger unit, iii) a distributed constraint checker (CC), iv) an exception-handling unit and v) a threshold-update unit. Below, we discuss each unit in detail, along with the steps of the topology construction. Finally, a minimalistic *topology generator network* is used to control the distributed construction of new topologies.

A **traffic directory** is placed at each router so that it can track all those source-destination pairs that have it as destination. HiROIC utilizes this information to determine the FCP set at the end of each epoch. This solution requires packets to carry their source node IDs, but this is a common practice in commercial NoC designs . To approximate a centralized, global computation of the FCP set (see Section III), we compare entries in the traffic directory against a threshold value (*high-traffic threshold*, or HT_{th}), stored at each router and managed dynamically through the threshold-update unit. All entries in the directory are compared against this threshold value: if the number of packets received from a given source is higher than the threshold, than the corresponding pair is included in the FCP set. The complete FCP set corresponds to the union of each router's FCP set – although we never compute this union and we keep the set distributed through all routers. The ideal HT_{th} depends on the epoch's length and on the communication load of the application, and thus it must be adjusted dynamically.

Our experiments across a range of workloads show that source-destination pairs transferring more than $\sim$256 packets within a single epoch of 10,000 cycles, are well above any dynamically generated high-traffic threshold, irrespectively of network load. Thus, we set traffic directory entries to be 8-bits wide (note that this design parameter depends on epoch's length). In our experiments, we use a network with 64 routers, thus the traffic directory at each router consists of 64*(8+6) = 896 bits, where 8 bits store the number of packets received from a given source, and 6 bits represent the source router tag.

Constraint checker. It is not always possible to enable the shortest paths between all FCPs. This is because HiROIC's architecture is limited by: i) the number of available router ports, and ii) the flexibility provided by the glue logic binding links and ports. Therefore, a check is performed after enabling each link to determine that these constraints are not violated. If any constraint is violated for any port-link along the path of an FCP, then all the links bound for that path are released. Constraint checking is fairly straightforward: a port-link binding cannot be performed if the concerned port is already bound to another link.

Reconfiguration-trigger unit. A reconfiguration event involves procedures that require significant activity. Fortunately, applications do not change phase as quickly as our target epoch length (see Section III). Thus, we trigger reconfiguration only when the application's communication patterns have changed significantly, and the current topology no longer provides low-latency paths for the current FCP set. Since ours is a distributed solution, we monitor for communication pattern changes locally, and each router is capable of triggering a reconfiguration if it detects significant changes. Specifically, we check for the following two conditions: i) the set of FCP entries should have at least *three* new members compared to the last reconfiguration, and ii) the FCP set should contribute at least 50% to the total router's traffic. The first condition ensures that changing the topology will significantly perturb the system, while the second condition guarantees the existence of non-uniform traffic patterns. We have calibrated these decision parameters through design space exploration and by taking into consideration hardware implementation costs. (for instance 50% of total traffic can be checked with a shift-and-compare operation, while other fractions may require far more complex computations).

The **exception-handling unit** monitors i) network congestion and ii) number of FCP entries per router, and then uses this information to update the HT_{th} threshold value. Congestion is a limiting issue in irregular topologies at medium-to-high traffic because they fail to appropriately balance traffic. At the onset of congestion, the benefits of shorter paths with irregular topologies are diminished, as packets have to wait longer for free channels and buffers. Our analysis shows that, in these situations, a baseline topology (such as 2D mesh in our case) may better balance traffic, and thus leads to less congestion. HiROIC leverages a local congestion detection metric (maximum buffer occupancy - called BFM in [6]): the network is considered congested if the BFM value is above a certain threshold (we found 20 to strike a good balance in our evaluation setup). Upon detecting congestion, the exception-handling unit broadcasts a 1-bit signal, similar to the *disconnection signal*, to all the routers. Once again, the routers revert to their baseline topology on reception of this signal.

The exception-handling unit also provides feedback to the *threshold-update unit*. By maintaining a suitable HT_{th} value, HiROIC ensures that the topology is reconfigured for an optimal number of FCP entries. Our experiments show that reconfiguring the topology for up to 50 FCP entries, results in increasing improvements over the baseline topology. Beyond 50 FCPs, the topology reconfiguration algorithm is unable to provide optimized paths for all FCPs, and this results in diminishing returns. However, controlling precisely the number of FCP entires in the network requires collection and sorting of usage statistics at a central node. For a distributed and fast implementation like ours, we control the number of FCP entries approximately by using only local criteria. The expectation is that by controlling the number of FCP entries per router, the number of FCP entries can be controlled globally. Also, by using a consistent HT_{th} value throughout the network, HiROIC ensures that all selected FCPs have higher usage than all other source-destination pairs. We experimentally determined that if the number of FCP entries at any router is more than four, then topology reconfiguration will provide little benefit due to excessive FCP entries. Consequently, increasing HT_{th}, which in turn reduces the number of FCP entries, will likely result in performance improvements. In practice, we want to keep the number of FCPs in each router within a close range of 4-5 pairs, and we keep pushing the threshold until we stabilize on that size. Such a simple scheme provides a suitable trade-off between topology optimality and simplicity of the hardware implementation. Thus, the exception-handling unit at any router broadcasts a "HT_{th} increase" flag globally, if the number of locally selected FCP entries are greater than four.

A **threshold-update unit** is deployed at each router and controls the high-traffic threshold by monitoring the broadcasts from the exception-handling units. Notice that HT_{th} affects directly the FCP set: the higher the threshold the fewer the pairs included in the FCP set. In addition, a suitable HT_{th} depends on the application: a communication-light workload will have a lower suitable HT_{th}, compared to a communication-heavy workload. Therefore, HiROIC can adapt to the needs of the application by tuning this value.

Increases to the high-traffic threshold value arise when there are changes in the workload communication density, from light to heavy, and they are triggered by the exception-handling unit. The opposite trend, traffic becoming lighter, is detected when HiROIC does not observe any exception for a number of consecutive epochs, indicating that we should include more pairs in the FCP set. We set the parameters of our algorithm by sweeping a range of values and selecting the most fitting ones: we initialize the HT_{th} at a moderately high value of 96 packets to prevent over fitting. HiROIC gradually adapts to the network's demands by varying HT_{th} in quanta of 8 packets. If we also do not observe an exception for 10 consecutive epochs (indicating a decrease in traffic), then we decrement HT_{th} to optimize for more FCPs.

The **topology-generator network** serves two purposes: i) transfer of control between routers during the building of a new topology, and ii) network-level broadcasts to notify all routers about exceptions detected locally at any router. For the former, the topology generator network employs 1-bit links per channel of the underlying NoC to communicate port-link binding signals. For the latter, the topology generator network deploys two 1-bit wires organized as a unidirectional ring that visit all routers in the NoC. Any router can broadcast exception detection flags using these wires. The first 1-bit wire is used to broadcast the "disconnection" or "congestion" flag. Both flags are identical in their effects and therefore they can be broadcasted on the same wire. The second 1-bit wire is used to broadcast the "HT_{TH} increase" flag. All routers have distributed controllers that snoop and forward these broadcasted flags.

VI. Physical Topologies

The key idea behind HiROIC is to emulate a router with more links than available ports. We refer to a particular arrangement of physical links and ports, independently of any binding, as a *physical topology*. In physical topologies, links are available in accordance to a high-radix topology (*e.g.*, 3D torus), while ports are those of a low-radix router (*e.g.*, 2D mesh routers). Naturally, HiROIC's efficacy greatly depends on the arrangement of links and ports within the physical topology. In our evaluation, we consider two such topologies: both use routers with only four ports (plus a port connecting the local node), as in a mesh. We argue that due to the similar router structure and considering power-gating of idle links, both topologies have power and area characteristics similar to a 2D mesh network. Therefore, all performance analysis is against a 2D mesh topology. Notice that a traditional 2D mesh has a one-to-one binding between ports and the links, as depicted in Figure 5(a). We implement HiROIC with the following physical topologies:

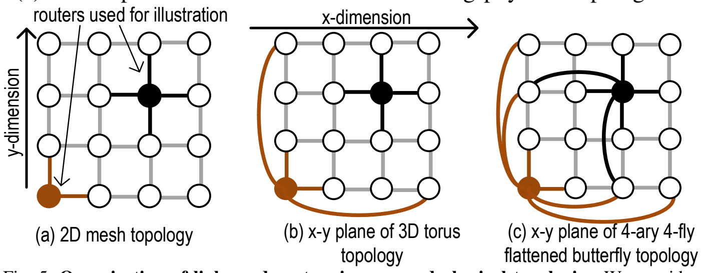
routers used for illustration
x-dimension
y-dimension
(a) 2D mesh topology
(b) x-y plane of 3D torus topology
(c) x-y plane of 4-ary 4-fly flattened butterfly topology

Fig. 5: **Organization of links and routers in proposed physical topologies.** We consider two topologies for links: a 3D torus and a flattened butterfly. Routers are organized as in a 2D mesh. For simplicity of illustration, the figure shows the x- and y- dimension connections only for the bold colored routers. 3D torus routers have two connections in each dimension, while a 4-ary 4-fly flattened butterfly has three per dimension.

Adaptive 3D Torus. Because of its high-radix(6) routers, the average hop count between the nodes of a 3D torus is substantially lower than that of a 2D mesh. We propose a HiROIC-enabled adaptive 3D torus physical topology that organizes links as in a 3D torus, while maintaining radix-four routers.

Adaptive Flattened Butterfly. A flattened butterfly further reduces the average hop count compared to 3D tori. For our 64-node NoC, a 4-ary 4-fly flattened butterfly arranges the routers in three dimensions, with direct links between routers on the same (x, y), (y, z) or (z, x) dimensions, as shown in Figure 5(c). Our second physical topology is an adaptive flattened butterfly (radix=9) topology with radix-four routers.

All routers are augmented with glue logic that incorporates multiplexers to bind ports and links at runtime. The size and the number of multiplexers depend on the number of links a particular router port can bind to. To understand the trade-off between multiplexer overhead and binding flexibility, consider the scenario in which an adaptive 3D torus router is provided full flexibility to connect any port to any link. In such a scenario, each output port can connect to any of the six links, requiring six 4:1 multiplexers. On the other hand, each input port can be bound to any of the four incoming links, resulting in four 6:1 multiplexers. Such a degree of multiplexing typically leads to layout challenges, in addition to area and power overhead.

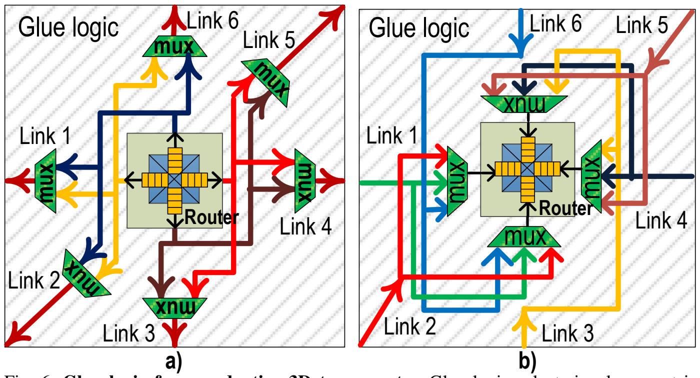

Fig. 6: **Glue logic for an adaptive 3D torus router.** Glue logic select signals are set in accordance to the new port-link bindings.

We experimentally concluded that increasing port-link binding flexibility beyond a certain degree is not beneficial, considering the extra logic overhead. We therefore decided to limit the number of links a port can choose from: in our adaptive 3D torus topology, each output port can connect to one of three output links (one in each dimension). Of course, this restriction prohibits some port link bindings: with reference to the example of Figure 6(a), the router can only make two connections from the set of links 1, 2 and 6. With these restrictions, each output link can connect to one of two ports, and therefore six 2:1 multiplexers are sufficient for the output glue logic. For the input glue logic, each input port can bind to one of three links, resulting in four 3:1 multiplexers, as shown in the Figure 6(b). Similarly for the adaptive flattened butterfly topology, we restrict each router's output port to connect to one of six output links (two in each dimension). This restriction results in the addition of one 6:1 and three 3:1 multiplexers for the input glue logic, while using six 2:1 and three 4:1 multiplexers for the output glue logic.

The restrictions on port-link bindings act as constraints in the topology reconfiguration algorithm (Section V). First, each router can never connect to more links than its number of ports (four in our physical topologies). Additional constraints are specific to the topology and arise from the limited flexibility of the glue logic, as described above.

As it is common in constraint satisfaction problems, some input conditions might not result in a solution satisfying all the constraints. In order to improve the chances of a valid solution, we relax the constraints that are not vital for correct functionality. Particularly, our reconfiguration algorithm accepts topologies with routers having fewer than four enabled ports. We are therefore able to achieve fully-connected and valid topologies on 99.5% of the reconfiguration events, as reported in Section IV. Obviously, these relaxed constraints can result in routers with fewer active ports, and thus they can increase the average hop count. To estimate the effect of constraint relaxation, we conducted Monte Carlo simulations to determine the fraction of routers that do not bind all of their four ports upon a topology reconfiguration. Table I indicates that, even after binding ports and links for 50 FCPs on an adaptive 3D torus, 94% of the routers bind all their ports. This result provides empirical evidence that our relaxed constraint has minimal impact on port utilization.

(a) Adaptive 3D torus

Number of pre-selected links	% of routers completely bound
10	96.46
20	95.67
30	95.18
40	94.65
50	94.36

(b) Adaptive flattened butterfly

Number of pre-selected links	% of routers completely bound
10	97.71
20	97.18
30	96.53
40	96.32
50	95.68

TABLE I: **Topology configurations allowing unbound router ports – analytical study.** The Monte Carlo analysis uses up to 50 FCPs, then binds the remaining ports. The table shows that ∼95% routers still bind all their 4 ports.

VII. EXPERIMENTAL RESULTS

We evaluated HiROIC on a cycle-accurate trace-driven multi-core simulator [5]. Table II shows the characteristics of the processors and the NoC we evaluated. We ran all experiments considering a 64-cores CMP as a baseline. The application traces were obtained using the PIN instrumentation tool. The simulator further incorporates a detailed model of the NoC with 3-stage pipelined routers. We implemented our scheme on top of an adaptive 3D torus and an adaptive flattened butterfly, as discussed in the previous section. All our comparisons are with respect to a baseline 2D mesh. Finally, we use an optimized version [15] of the up*/down* [16] routing algorithm for the HiROIC-enabled NoC, while the baseline system uses XY routing.

(a) Processor @2GHz

Cores	2-wide fetch/commit 64-entry ROB
coherence	4-hop MESI, 64B block
L1 cache	Private: 32KB/node ways:4 latency:2
L2 cache	Shared: 256KB/node ways:16 latency:6
Memory	Distributed: 1GB/bank banks:4 latency:160

(b) Network @2GHz

Topology	8x8 mesh, 128 bit links
Pipeline	3-stage VC flow ctrl
VCs	4 VCs/port, 8 flits/VC
Routing	up*/down*,XY
Routing-Update	Ariadne [1]: new up*/down* routes
Workload	multi-programmed: SPEC CPU 2006
Simulation	10M cycles

TABLE II: **Experimental CMP: configuration of processor and network.**

A. *Synthetic Traffic*

Our first set of experiments injects the NoC with synthetic normal random traffic. Normal random traffic is the most adverse traffic pattern for topology optimization, since it does not create any imbalance on the network. Since all nodes share similar amounts of traffic, the FCP set should ideally be empty and topology reconfiguration should never be triggered. However, since we use a distributed approximation for our algorithm, we may observe some reconfiguration invocations. For the set of decision parameters discussed in Section V, however, we do not observe any reconfiguration invocations in this experiment, leading to our adaptive topologies behaving exactly as a 2D mesh.

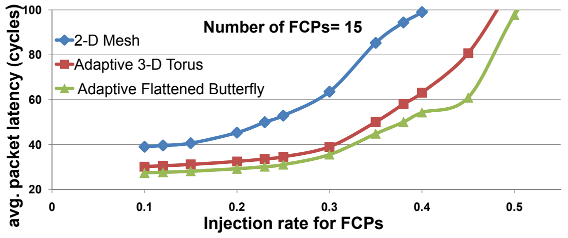

Fig. 7: **Average network latency under directed traffic.** The plot compares the average network latency for three topologies under directed traffic with increasing injection rate for the FCPs. HiROIC provides low-latency paths between FCPs, resulting in significant overall latency improvements.

HiROIC is expected to provide latency improvements when some source-destination pairs transfer more traffic than others. We created synthetic *directed traffic* to gain more insights into the strengths and limitations of our scheme. Our synthetic directed workloads consist of 20 phases, each of which lasts 50 epochs and has a number of frequently communicating pairs (FCPs). The new FCP set is randomly selected after each phase. Correspondingly, HiROIC triggers a new reconfiguration after each phase change. Other network nodes produce traffic at low injection rate (0.005 flits/node/cycle). Figure 7 compares the average latency of the topologies under consideration with directed traffic using 15 FCPs. On the x-axis we sweep the injection rate for the FCPs. To compare latency improvements, we define three traffic load levels for the FCPs: low, medium and high. Low traffic corresponds to 0.1 flits/node/cycle, and it is the lowest injection rate used in our experiments. The medium and high injection rates are defined as the injection rates where the network latency for the 2D mesh is 1.5× and 2× that of the low-load latency. We observe that the latency improvement over 2D mesh for adaptive 3D torus is 22.7%, 29.3% and 36.9% for low, medium and high injection rates, respectively, while the corresponding latency improvements for adaptive flattened butterfly are slightly better at 30.8%, 35.6% and 37.8%. This experiment proves HiROIC's potential in providing significant reduction in network latency in the presence of traffic imbalance.

In order to study the limitations of HiROIC, we swept the number of FCP entries for a fixed directed load of 0.3 flits/node/cycle (corresponding to medium load). Since irregular topologies realized by HiROIC suffer from congestion at medium-to-high traffic, our scheme reverts back to a 2D mesh topology upon congestion detection. Therefore, increasing the number of traffic-heavy FCPs beyond a certain point, should result in ineffective topology reconfigurations. Figure 8 shows that beyond 25 FCPs, HiROIC's improvements over 2D mesh start diminishing. However, the optimal number of FCP entries varies depending on the network load: a heavily loaded network saturates HiROIC's benefits with a smaller number of FCPs.

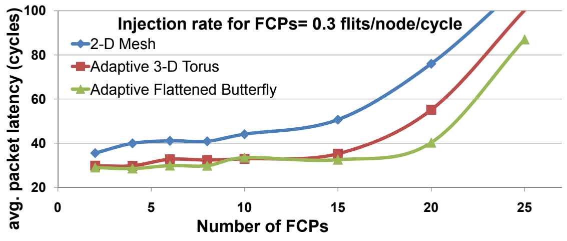

Fig. 8: **Average network latency with varying number of FCPs.** The plot compares the average network latency under medium-load directed traffic: HiROIC's latency improvements start diminishing beyond 25 FCPs because of network congestion.

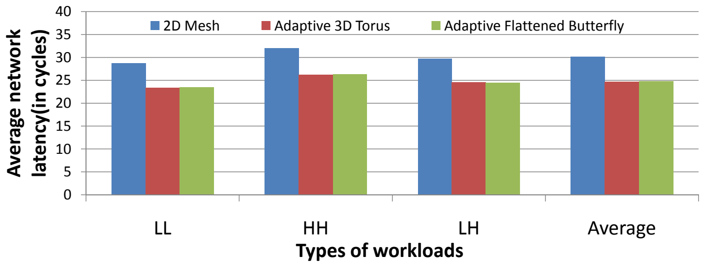

Fig. 9: **Average network latency under multi-programmed workloads.** The results are presented for 2D mesh, adaptive 3D torus and adaptive flattened butterfly topologies under three different types of workloads. HiROIC is most effective for workloads in the *LH* category due to high traffic imbalance.

B. Multiprogrammed Workloads

We also evaluated our proposed scheme with a set of multiprogrammed workloads consisting of 35 applications from the SPEC CPU 2006 benchmark suite. The experiments were conducted across 60 multiprogrammed workloads, with each workload consisting of 15 copies each of 4 unique applications. The studied applications exhibit a wide range of cache misses per kilo instructions (MPKI) values: the MPKI metric directly correlates to the amount of traffic sent through the NoC. Some workloads use applications with similar MPKI values causing all cores to inject similar amount of traffic on the NoC. We further divide such workloads into two categories: the *LL* category workloads use applications with *low* MPKI, while the *HH* workloads use applications with *high* MPKI. We also use imbalanced workloads, in which the MPKI values among the applications differ substantially, and group them under *LH* category. We run each workload for 1,000 computational epochs and we noted that, on average, HiROIC triggered 142 reconfigurations during each execution.

Figure 9 compares the network latency of a 2D mesh against the HiROIC-enabled adaptive 3D torus and adaptive flattened butterfly under multiprogrammed workloads. Since HiROIC can optimize the hop count only for a subset of communication paths, it provides the highest latency improvements for high imbalance workloads, *i.e.*, workloads in the *LH* category. For this category, the average latency reduction over 2D mesh is 21.4% and 21.2% for adaptive 3D torus and adaptive flattened butterfly, respectively. HiROIC also provides good improvements for workloads in the *LL* category because network transmissions are scarce in such workloads, and only a small subset of applications produce significant traffic within a given computational epoch (in contrast to all applications producing traffic all the time). Therefore, HiROIC optimizes the topology for this subset and reduces the average network latency by 18.5% and 17% for adaptive 3D torus and adaptive flattened butterfly topology. Finally, we observe the smallest gains for workloads in the *HH* category, as most nodes generate heavy traffic, leading to a larger than optimal FCP set. For workloads in the *HH* category, the average latency improvement for adaptive 3D torus and adaptive flattened butterfly over 2D mesh is 17.0% and 16.7%, respectively.

Our evaluations with multiprogrammed workloads do not yield results as promising as the directed traffic evaluation of Section VII-A. The primary reason for this is the organization of the underlying CMP system. Our baseline CMP uses a shared and distributed L2 cache architecture, and therefore L1 cache misses are uniformly distributed over the entire CMP. As a result of this distribution, the majority of the traffic in the NoC is uniform, and HiROIC is not able to optimize communication paths aggressively, leading to sub-optimal results. However, with a growing adoption of application-specific accelerators on-chip [4], only a small subset of which will be active at any point in time, we expect a greater imbalance in communication for future architectures. The scenario is expected to be similar to our synthetic directed traffic experiments from Section VII-A, and thus we expect HiROIC to provide even better benefits in future architectures.

C. Area and Power Overhead

We implemented each HiROIC component in Verilog and used a publicly available implementation of a 2D mesh router [3]. We synthesized the HiROIC components and the unmodified router individually, using Synopsys Design Compiler and the ARM ARTISAN 45nm library. The logic area overhead of the HiROIC components for an adaptive 3D torus router, relative to the unmodified router area are: i) multiplexers in the glue logic (1.1%), ii) traffic directory (2.2%), iii) topology-generator network (0.5%). Thus, we accrue approximately 4% overhead over a 2D mesh router, in contrast with the 38% of a high-radix 3D torus router. We assume that the Ariadne-style route-reconfiguration functionality, including routing tables, is already available at each router for fault-tolerance. If not, Ariadne can be implemented at $<2\%$ overhead [1]. For a flattened butterfly router, the area overhead is slightly higher because of the additional multiplexers in the glue logic. However, the overall area overhead is still small ($\sim$4%) and should not drive the selection of the physical topology.

The main sources of power overhead in HiROIC are the glue logic and the additional link wires in the physical topology. On the other hand, HiROIC saves power by reducing the hop count of many packets – all those using an FCP – and we believe that these savings are larger than the additional power costs.

VIII. Conclusion

HiROIC provides performance similar to high-radix ($>$ 5 ports) NoC topologies using resources comparable to low-radix topologies ($<=$ 5 ports) by optimizing for critical high-volume communication paths at runtime. In HiROIC, links are deployed abundantly for rich connectivity as in high-radix topologies, while the number of router ports is kept low. Router ports bind to links at runtime in accordance to a distributed traffic analysis heuristic implemented at each router. Our experiments with multi-programmed workloads on a 64-node CMP, show that HiROIC reduces average network latency by 19% compared to an area- and power- comparable mesh. When using non-uniform synthetic traffic, the latency reduction is in the 30-38% range.

Acknowledgements: This work was partially supported by NSF grant #0746425 and CFAR, within STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco. ARIADNE: Agnostic reconfiguration in a disconnected network environment. In *Proc. PACT*, 2011.
[2] J. Balfour and W. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proc. ICS*, 2006.
[3] D. Becker. Efficient microarchitecture for network-on-chip routers, PhD thesis. 2012.
[4] Y.-T. Chen, J. Cong, M. Ghodrat, M. Huang, C. Liu, B. Xiao, and Y. Zou. Accelerator-rich cmps: From concept to real hardware. In *Proc. ICCD*, 2013.
[5] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Application-aware prioritization mechanisms for on-chip networks. In *Proc. MICRO*, 2009.
[6] R. Das, S. Narayanasamy, S. Satpathy, and R. Dreslinski. Catnap: energy proportional multiple network-on-chip. In *Proc. ISCA*, 2013.
[7] M. Faruque, T. Ebi, and J. Henkel. Configurable links for runtime adaptive on-chip communication. In *Proc. DATE*, 2009.
[8] B. Fu, Y. Han, J. Ma, H. Li, and X. Li. An abacus turn model for time/space-efficient reconfigurable routing. In *Proc. ISCA*, 2011.
[9] J. Kim. Low-cost router microarchitecture for on-chip networks. In *Proc. MICRO*, 2009.
[10] T. Krishna, W. Chen, C-H.and Kwon, and L.-S. Peh. Breaking the on-chip latency barrier using SMART. In *Proc. HPCA*, 2013.
[11] O. Lysne, J. Montañana, J. Flich, J. Duato, T. Pinkston, and T. Skeie. An efficient and deadlock-free network reconfiguration protocol. *IEEE Trans. Computers*, 57(6), 2008.
[12] M. Palesi, R. Holsmark, S. Kumar, and V. Catania. *IEEE Trans. Parallel and Distributed Systems*, 20(3), 2009.
[13] R. Parikh and V. Bertacco. uDIREC: unified diagnosis and reconfiguration for frugal bypass of NoC faults. In *Proc. MICRO*, 2013.
[14] L.-S. Peh and W. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. HPCA*, 2001.
[15] J. Sancho, A. Robles, and J. Duato. An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Trans. Parallel and Distributed Systems*, 15(8), 2004.
[16] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Trans. Selected Areas in Communication*, 9(8), 1991.
[17] M. Stuart, M. Stensgaard, and J. Sparsø. The ReNoC Reconfigurable Network-on-Chip: Architecture, Configuration Algorithms, and Evaluation. *ACM Trans. Embed. Comput. Syst.*, 10(4), 2011.