Resilient and Power-Efficient Multi-Function Channel Buffers in Network-on-Chip Architectures

Dominic DiTomaso, *Student Member, IEEE*, Avinash Karanth Kodi, *Senior Member, IEEE*, Ahmed Louri, *Fellow, IEEE*, and Razvan Bunescu

Abstract—Network-on-Chips (NoCs) are quickly becoming the standard communication paradigm for the growing number of cores on the chip. While NoCs can deliver sufficient bandwidth and enhance scalability, NoCs suffer from high power consumption due to the router microarchitecture and communication channels that facilitate inter-core communication. As technology keeps scaling down in the nanometer regime, unpredictable device behavior due to aging, infant mortality, design defects, soft errors, aggressive design, and process-voltage-temperature variations, will increase and will result in a significant increase in faults (both permanent and transient) and hardware failures. In this paper, we propose *QORE*—a fault tolerant NoC architecture with Multi-Function Channel (MFC) buffers. The use of MFC buffers and their associated control (link and fault controllers) enhance fault-tolerance by allowing the NoC to dynamically adapt to faults at the link level and reverse propagation direction to avoid faulty links. Additionally, MFC buffers reduce router power and improve performance by eliminating in-router buffering. We utilize a machine learning technique in our link controllers to predict the direction of traffic flow in order to more efficiently reverse links. Our simulation results using real benchmarks and synthetic traffic mixes show that QORE improves speedup by 1.3× and throughput by 2.3× when compared to state-of-the art fault tolerant NoCs designs such as Ariadne and Vicis. Moreover, using Synopsys Design Compiler, we also show that network power in QORE is reduced by 21 percent with minimal control overhead.

Index Terms—Network-on-chips, fault tolerance, power-efficiency

1 Introduction

As transistor technology scales down to the sub-nanometer region, integrated circuit (IC) designs are moving towards chip multiprocessors (CMPs), integrating hundreds of smaller processing elements or cores on a single chip. As the number of cores continue to scale and conventional on-chip bus-based communications approach their limits, architects were driven to consider other scalable communication strategies. Network-on-Chips (NoCs) have emerged as the de facto communication paradigm by offering scalability through a modular design [1], [2]. In NoCs, segments of links are connected via routers in order to overcome global wire delays and scalability requirements. However, the combination of links and routers incur a power and area expense which adversely affects NoC performance. Extensive power optimization techniques have been used to mitigate the NoC power consumption. The NoC of Intel's 80-core TeraFlops chip [3] consumes 28 percent of the total tile power using simple cores, whereas the NoC in the more recent Intel 48-core SCC [4] consumes 10 percent of the tile power using regular cores. Power optimizations of the NoC fabric is a critical piece of the puzzle to sustain and continue the drastic growth in CMP performance.

A typical NoC hardware consists of routers and communication channels connecting the routers. The router is comprised of input/output ports, buffers, routing logic, and a crossbar connecting input ports to output ports for packet routing. Research has shown that router buffers are responsible for 46 percent of router power [5] and 30 percent of router area [6]. This has motivated architects to implement buffer optimization techniques such as elastic buffering [7], [8], [9] and bufferless routing [10], [11], [12]. By completely eliminating buffers and implementing bufferless routing, recent work has reduced the average network energy by 40 percent [10]. Buffers have been moved from the router to the channels by replacing repeaters on the channel with either flip-flops called elastic buffers [8] or tri-state repeaters called channel buffers [9]. These channel buffers can store packets when the register buffers are congested or propagate data forward when necessary, thereby mitigating power and area penalties associated with router buffers.

Since the buffers have been moved to the channels, hard errors in the channel buffers can cause complete failure, impeding communication between routers. Researchers have been tackling channel failure in NoCs [13], [14], [15], [16], [17], [18], [19], [20]. Recently, the fault tolerant Ariadne network [13] overcame channel faults by reconfiguring packet routing to move around failures. Using up*/down* routing and a series of flag broadcasts, the two unidirectional links between routers were dynamically assigned as

- *D. DiTomaso, A.K. Kodi, and R. Bunescu are with the Department of EECS, Ohio University, Athens, OH 45701. E-mail: {dd292006, kodi, bunescu}@ohio.edu.*
- *A. Louri is with the Department of Electrical and Computer Engineering, University of Arizona, Tuscon, AZ 85721. E-mail: louri@ece.arizona.edu.*

Manuscript received 27 May 2014; revised 30 Jan. 2015; accepted 1 Feb. 2015. Date of publication 5 Feb. 2015; date of current version 11 Nov. 2015. Recommended for acceptance by R. G. Melhem. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2015.2401013

up or down to create a tree network that avoid faulty channels. Another fault tolerant NoC called Vicis [14] routed around faults and placed turn restrictions at routers to avoid deadlocks. The BulletProof architecture [21] concentrates on the router (not the channels) and provides efficient fault tolerance schemes for routers to overcome transient and permanent faults.

With the increasing number of cores, NoCs must manage the communication demands, especially when faults are present. Since NoCs are designed to handle peak traffic loads, many communication channels can go under-utilized when network load is high or the workload is unbalanced. Recent research on NoC performance has tackled above mentioned problems using techniques such as reversibility or coding schemes [22], [23], [24], [25], [26], [27]. Hesse et al. propose a bandwidth-adaptive router (BAR) that aims to take advantage of these under-utilized links with bidirectional, adaptive channels [23]. These bidirectional channels adapt channel bandwidth at a fine-granularity according to network traffic demands. Research has shown that channel reversibility can achieve higher throughput and lower average packet latency in NoCs.

In this paper, we propose *QORE*—a fault tolerant network-on-chip architecture with power-efficient Multi-Function Channel (MFC) buffers. QORE can lower power with channel buffers, improve performance through reversible links, and improve fault tolerance through redundant links. The key component in QORE are the MFCs which have multiple functionalities: on-demand data storage, on-demand forward data propagation, and backward data propagation. On-demand data storage enables communication channels to act as buffers and store data when the network load is low and function as repeaters when the network load is high. Therefore, MFCs have five possible states: forward propagation, backward propagation, forward buffer, backward buffer, and flush. Using multiple links with the MFCs and the associated control blocks, QORE can improve both performance and fault tolerance. The multiple links between routers can provide data with redundant paths in case of faults. Increasing the number of links between routers can improve fault tolerance; however, the bandwidth of the links decreases. Therefore, there is a trade-off between fault tolerance and performance as the number of links varies. We evaluate this trade-off by varying the number of links from two to eight. Additionally, we use machine learning (ML) techniques to accurately determine how to reverse the various links in QORE. The proposed QORE architecture attempts to address three issues of power, performance and fault-tolerance in a cohesive manner. The major contributions of this work over our prior work [28] include:

- *Multi-Function Channel Buffers.* We design channel buffers that can dynamically function as (1) forward repeaters, (2) backward repeaters, (3) forward buffers, and (4) backward buffers. This enables significant power reduction with no router buffers while enabling circuits to both improve performance and reliability. We evaluate the effect of channel buffers in terms of power compared to designs with conventional register buffers.

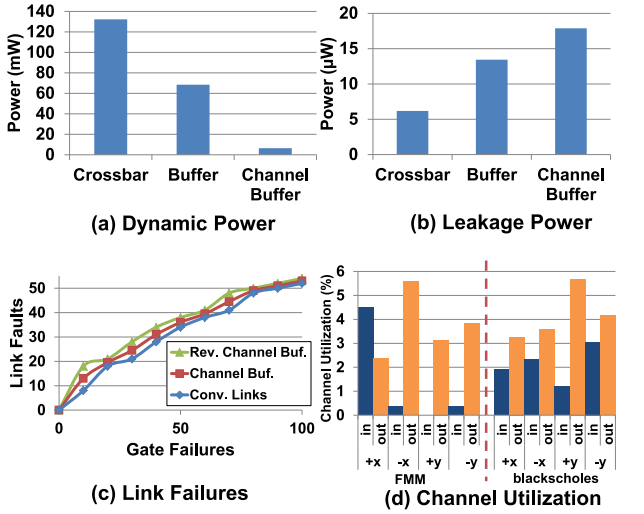


Fig. 1. (a-b) Dynamic and Leakage power of buffers, (c) link failures with and without channel buffers, and (d) link utilization of a router.

- *Direction of Traffic Prediction.* We propose to use a machine learning technique to predict the traffic flow on our links which can be used for more accurate link reversing. Experimental results show that a decision tree (DT) predicts the direction of the traffic with higher accuracy on average than a predictor based on thresholded link utilization.
- *QORE with Varying Links.* We vary the number of inter-router links in QORE from two to four to eight in order to study the effect on the network performance and fault-tolerance trade-off.

2 Motivation

In this paper, we focus on two important concerns in NoCs while also maintaining network performance: high power dissipation and declining reliability. As previously mentioned, buffers consume a major portion of the router power. This has been the key concern that has motivated researchers to develop novel ideas such as bufferless networks [10], [11], dynamic VC allocation [29], and elastic or channel buffers [8], [9]. Buffers consume significant dynamic power when traffic load is high as well as static power due to leakage. Fig. 1a, 1b shows the total power breakdown (in mW) for a 5×5 router from Synopsys Design Compiler using the TSMC-LPBWP 40 nm technology library with a nominal supply voltage of 1.0 V and an operating frequency of 2 GHz. The dynamic power breakdown in Fig. 1a shows that buffers consume 33 percent of router power (buffers+crossbar). With the same amount of buffer space, channel buffers can lower dynamic power by 90 percent. Fig. 1b shows the leakage power breakdown of the router components in μW. As shown, the leakage power of the buffers consume 68 percent of the total router leakage power (buffers+crossbar). Channel buffers dissipate more leakage power than the register buffers; however, this increase is compensated by the very low dynamic power. Therefore, one possible alternative to the high power register buffers in NoCs are channel buffers.

The next major concern in NoCs is reliability. The extreme shrinking of transistor feature sizes has made

NoCs vulnerable to failures and data corruption. To examine the number of link faults in an NoC, a fault model was used which was similar to the model used in [13] in which a router design consists of 20,413 gates. Faults were injected randomly and weighted by the size of the gates. Therefore, gates with a larger number of transistors have a higher probability of failing. Fig. 1c shows the number of faulty links caused by gate failures for reversible channel buffers (explained in Section 3), non-reversible channel buffers, and conventional links without channel buffers. Non-reversible channel buffers are less reliable than conventional links due to the extra two transistors added to each link. Reversible channel buffers are even less reliable because of the eight additional transistors as explained in Section 4.1. Therefore, robust fault tolerant techniques are even more critical when using channel buffers.

In addition to power and fault tolerance, high network performance is another concern in NoCs. Looking at a NoC router, the amount of traffic entering and leaving the router will be similar when averaged across the whole application. However, due to dynamic traffic patterns in NoC applications, there will be period of time where the majority of traffic will be either entering or leaving the router. This unbalanced traffic can cause certain links to become under-utilized during certain epochs. Fig. 1d shows the link utilization of a router in a 64 core network for two real applications. Each side of the router (+x, −x, +y, and −y) has a link going in and out. For the applications FMM and black-scholes [30], [31], many links are under-utilized, thereby, wasting bandwidth. For example, on the +x side of the router, the "in" channel utilization is approximately double the "out" channel utilization. On other links, the "in" utilization is much lower than the "out" utilization. Using reversibility, links can change direction providing bandwidth where needed. Channel buffers can reduce dynamic power while marginally increasing leakage power; and reversible channel buffers could maximize resource utilization and improve execution time, but would need fault tolerant techniques to overcome the higher fault rates observed in channel buffers.

3 Multi-Function Channel Buffers

In this section, we will explain the circuit and implementation details of our proposed MFC buffers. Channel buffers have been shown to eliminate router buffer power by moving storage to the channels with the side benefit of reducing the area overhead with marginal performance penalty [8], [9]. In this work, we uniquely modify the previously proposed channel buffers to function as bidirectional channel buffers with similar advantages of reduced power while providing on-demand storage. Fig. 2a shows two physical channels with four channel buffer stages per channel. The inset shows a conventional channel buffer which uses four transistors and a release (rel) control line to store or propagate packets in one direction. The working of channel buffers to either store or propagate packets based on router congestion and receive signals via a control block has been discussed previously [9]. The proposed reversible channel buffer circuit is shown in Fig. 2b. By adding eight transistors to

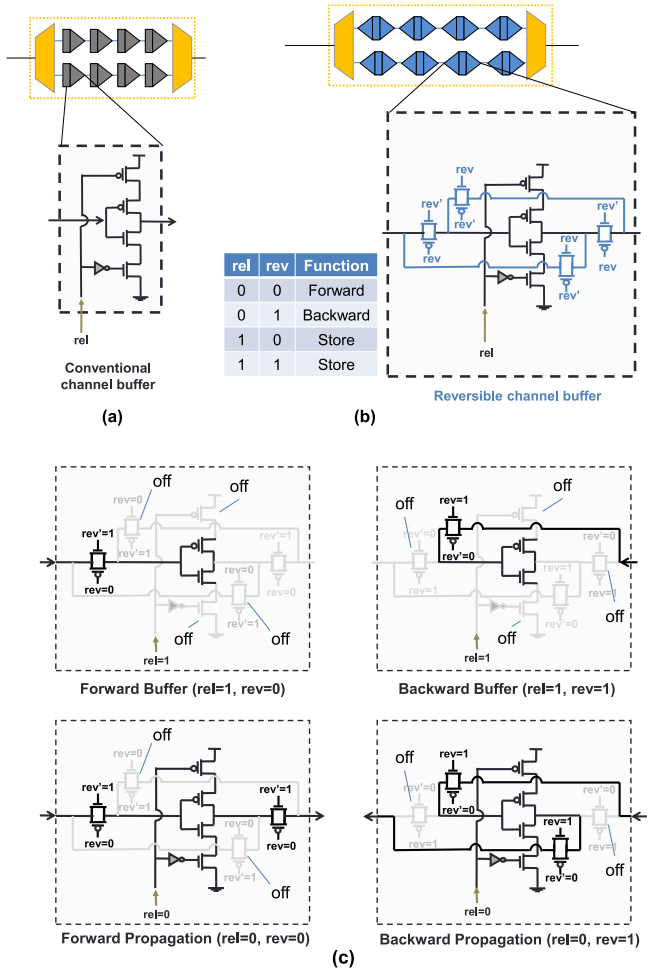


Fig. 2. (a) Conventional channel buffer, (b) our reversible channel buffer, and (c) storage and propagation for both forward and backward links.

act as four transmission gates, the channel buffers can propagate packets in both directions in addition to storage. The four transmission gates are controlled by the reverse signal (rev) sent from the router. A table showing all possible functions of the reversible channel buffer based on the inputs rel and rev are also shown in Fig. 2b. Fig. 2c shows various combinations of reversible channel buffer functionalities; either as on-demand storage or repeater, and with data propagating either in forward or backward directions.

- *Forward Buffer*. When $rel = 1$ and $rev = 0$ data can be stored in the forward direction (left to right). The data is cut off from V_{dd} and GND and the data is stored on the capacitance of the transistors.
- *Backward Buffer*. When $rel = 1$ and $rev = 1$ data can be stored in the backward direction (right to left). Again, the data is cut off from V_{dd} and GND and the data is stored on the capacitance of the transistors.
- *Forward Propagation*. When $rel = 0$ and $rev = 0$ data can propagate forward. The transistors connected to V_{dd} and GND are enabled to allow propagation and the forward propagation transmission gates are also enabled.
- *Backward Propagation*. When $rel = 0$ and $rev = 1$ data can propagate backward. Again, the transistors

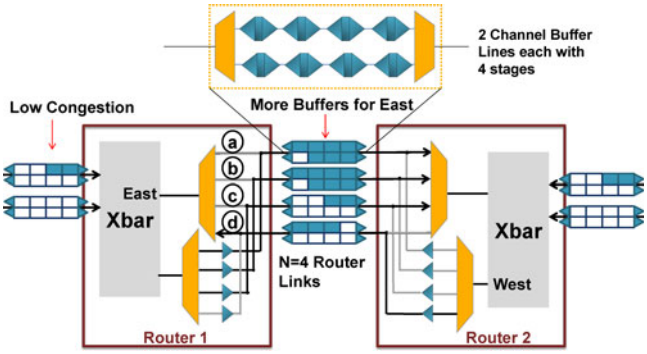


Fig. 3. QORE's four reversible router links each consisting of two channel buffer lines.

connected to V_{dd} and GND are enabled to allow propagation and now the backward propagation transmission gates are enabled.

We show four functions of our MFC for high network loads (forward and backward buffers) and for low network loads (forward and backward propagation). When our MFCs act as buffers, the capacitance of the transistors must be large enough to store the data for many cycles. We determined the discharge time of a channel buffer implemented with 130 nm transistors using the Virtuoso Analog Design Environment from the Cadence tools. The discharge time of the channel buffers is in the magnitude of milliseconds which corresponds to millions of clock cycles with a 1 GHz clock.

4 QORE Architecture

In this section, we will describe the QORE architecture including reversibility, the design and operation of the fault tolerant network, the details of the fault and link controllers (LC), the router microarchitecture, and proof of deadlock-freedom.

4.1 MFC without Faults

Conventional routers, that use virtual channels (VCs) and fixed connections between routers, can become a bottleneck if there is high traffic in any direction. To reduce the buffering bottleneck, QORE uses our reversible channel buffers to dynamically allocate buffers to adapt to traffic patterns. Fig. 3 shows the links between routers in QORE. In order to have the same amount of buffering as a conventional 4 VC/input router, we place a set of N=4 links between routers each consisting of two channel buffer lines. Each link consists of two channel buffer lines to alleviate HoL blocking [7]. Additionally, since QORE has more links between routers than the two links in conventional routers, we have reduced the bandwidth of our links for a fair comparison, as explained in the evaluation section. Therefore, the wire area overhead of QORE is equal to the conventional baseline networks. However, a designer can choose N to be a different number depending on system requirements. Each router link is reversible, allowing communication in both directions. However, the two channel buffer lines in each link will always be directed the same way. This will ensure that at any time, a packet will have at least two VCs to choose from, which in turn will alleviate HoL blocking. In QORE,

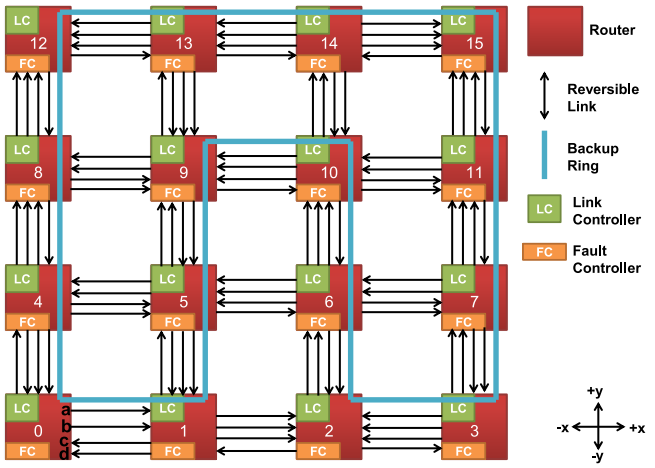


Fig. 4. Layout of QORE showing links configured to an arbitrary traffic pattern.

when there is high traffic in one direction, the links can change direction according to the traffic load, thereby increasing buffer space. For example, in Fig. 3, when there is high eastbound traffic, three links (a-c) can be allocated to the east direction while one link (d) remains in the west direction. The three east links can, therefore, use the under-utilized westbound buffers and provide more buffering for eastbound traffic. This additional buffering will relieve congestion at router 2 as well as router 1 and other upstream routers. Meanwhile, the one west link can still provide buffering for westbound traffic. As a result, both eastbound and westbound traffic can have ample buffering, thereby, decreasing packet latency. Therefore, reversing router links in QORE can reduce traffic bottlenecks caused by under-utilized links and buffers.

Determining which direction to allocate links is critical in QORE. Network traffic is measured using hardware counters to store the number of link traversals in each direction. A two-stage controller, which is detailed more in Section 4.3, is used to allocate links to the appropriate direction based on traffic demands. The first stage (link controller) of the controller uses the counters to determine which direction has the highest traffic called as the *"majority"*. The second stage (fault controller (FC)) will assign all but one link to the majority direction or allocate equal links to both directions if the link utilizations are similar. In the example in Fig. 3, each time a flit traverses links (a-d), both routers 1 and 2 will increment their counters. Since there is high eastbound traffic in this example, the link controllers will determine that the majority of the traffic is moving from west to east. At this point, the fault controllers in both router 1 and 2 will allocate the first three links (a-c) to the east and allocate link (d) to the west. If there are packets currently stored in the channel buffers when the reversing occurs, then these packets will be flushed out to escape VCs inside the downstream router.

4.2 MFC with Faults

QORE uses MFC buffers to overcome hard faults in the network. When a link in one direction is faulty, another link can reverse its direction to overcome this fault. Fig. 4 shows the overall layout of the QORE network for 16 routers and can be easily scaled to large numbers. The routers are connected

to each other in a grid-like fashion similar to a mesh network. However, instead of the two unidirectional links between routers as in a mesh, QORE has four, narrower reversible links between each router. Again, each reversible link consists of two channel buffer lines. Also, the links are narrower than the baseline links as explained in the evaluation section so there is no area overhead. The additional links create redundant paths between routers to improve both performance and reliability while avoiding HoL blocking. The link setup shown in Fig. 4 is arbitrary; each link can reverse in either direction depending on traffic demands. QORE also has a backup ring network [32] which is used when there are a large number of faults that potentially could isolate healthy routers. Each router has a link controller and a fault controller (Detailed in Section 4.3) that analyze link utilization and determine which links to reverse.

Each set of four links can handle up to three faulty links before using the backup ring. If a fault is detected in any of the links of a set, then the remaining non-faulty links will point in the directions specified by the LC and FC. For example, suppose the four links on the +x side of router 0 are initially setup as shown in Fig. 4 with two links facing east (E) and two links facing west (W). If faults are detected in both links a and b, then links c and d can overcome these faults by setting their directions to E and W, respectively. This will maintain connectivity between routers 0 and 1 so that packets can still be transmitted to both sides. If three of the four links fail then the fourth link can be used to communicate both ways since it is reversible. However, if all four links between two routers fail, then the backup ring network must be used. The backup ring network consists of two unidirectional rings, so that packets can traverse the shortest path, either clockwise or counterclockwise, to their destination. For example, if all four +x links of router 0 fail and the destination is router 5 then the packet will be routed on the ring network from router 0 to router 1, and so on up to router 5. Once a packet is on the ring network, it must stay on the ring network until it reaches its destination in order to avoid livelocks and deadlocks.

4.3 Link and Fault Controllers

In order to keep track of the status of each link, Link Status Tables (LSTs) are implemented in hardware. There are four LSTs per router in QORE; one for each set of links. The set of links on the right-side of a router are labelled as the +x links, links on the left are labelled $-$x links, etc. Each set of links has a LST containing information about the links. Each table has as many entries as links in each direction. In this paper, there are always four links in each direction. Hence, $n_{+x} = n_{-x} = n_{+y} = n_{-y} = 4$ and each table has four entries.

Each link in a specified direction has a unique identifier stored in the *Link Address* field. Whether the link is facing in towards the router or out away from the router is specified in the *Direction* field. This field will be read by the routing computation (RC) to determine valid routing paths and will be set up by the algorithm in the FC. The *Flit Count* data field stores the number of flit traversals on the link within the reconfiguration window, R_w. These counters are read by the LC to determine traffic demands. Each counter is incremented every time its corresponding link receives a flit and is decremented every time its corresponding link sends a flit. The *Faulty* data field stores whether or not the link is useable. This data field is read by the FC and RC. The field is set when its corresponding link detects a fault. Detection of faults can be done by implementing Built-In System Test (BIST) [15], [33]; however, fault detection is beyond the scope of this work. Finally, each table stores the total number of working links which is set each time a fault is detected.

The LC and FC are split into four independent blocks corresponding to each direction (+x, $-$x, etc.). The inputs of the LCs are the direction fields for each of the four router links. The output of the LCs indicates which direction (N=north, E=east, S=south, W=west, or B=both) the majority of the flits were traveling during the last R_w cycles. If the traffic was roughly equal (within Δ where Δ=5% of total flit traversals in this paper) then a B is output and an equal number of links will face in each direction. The LC output gives a good measure on the traffic demand so that link bandwidth can be properly allocated. The simple algorithm to determine the majority of the +x (px) links is shown in Algorithm 1. We will show how to improve this algorithm using machine learning in Section 4.4. At the end of R_w, the LCs total up the counts from their corresponding LSTs. Since the counters are incremented when a flit is received and decremented otherwise, a positive total would indicate the majority of the traffic is moving "West" for the set of +x links and a negative total would indicate more "East" Traffic. At the end, the counts in the LSTs are cleared for next R_w. The majority output is then fed to the FCs.

Algorithm 1. Link Controller and Fault Controller Pseudocode for +x (px) Links

```
// Link Controller
if(Enable){
  for(all links 0 to n_{+x} − 1)
    total_count = total_count + pxLnk[i].count;
  if(total_count ≫ 0)
    pxMajority = West;
  else if(total_count ≪ 0)
    pxMajority = East;
  else
    pxMajority = Both;
  clear_all_counts();
}
// Fault Controller
if(Majority of traffic is West){
  if(pxLnk.totalGood == 1)
    assign_one_link(West);
  else{
    assign_one_link(East);
    assign_remaining_links(West);
  }
} else if(Majority of traffic is East){
  // Same as above except interchange West and East
} else if(Traffic is similar in both directions){
  assign_half_links(West);
  assign_half_links(East);
}
```

The inputs for the FC are the majority signal, the total number of good links, and the fault status of each link. The

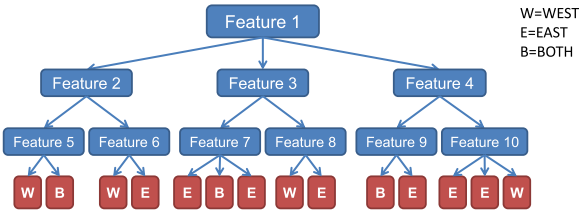


Fig. 5. An example decision tree.

FC determines the new directions for each link by outputting their link address and updating the direction field in the LST. The algorithm to determine the directions for the +x set of links is shown in Algorithm 1. If the LC determines that the majority is W, then the FC will assign a majority of the links to the W direction. The FC also tries to maintain connectivity by assigning at least one link to the opposite direction of the majority when possible in the $assign\_one\_link$ function. The remaining links are assigned to the majority direction in the $assign\_remaining\_links$ function. Both $assign\_one\_link$ and $assign\_remaining\_links$ functions are optimized so that the number of links that change directions is minimized. Therefore, with four links between routers, after FC at most two links will change direction. This minimizes the number of flits that must be flushed. When there is only one non-faulty link, then the FC must break connectivity and assign the link to the majority direction. However, this will cause starvation as packets cannot be sent in one direction. We resolve this by allocating 60 percent of R_w to the majority direction and reserve 40 percent of R_w to the opposite direction. We chose 60 percent because our simulation results showed that this value gave the best average performance over all the benchmarks.

4.4 Improving Link Controllers with ML

In order to improve our link controllers, we propose to use machine learning techniques to predict the traffic flow on the links. Our baseline link controllers only use link utilization from the previous time window to predict the traffic for the next time window. For example, if the majority of traffic was going east during the last reconfiguration window then the links are adjusted so that more bandwidth is given to the east direction for the next reconfiguration window. However, with dynamic traffic patterns the direction of the traffic may drastically change from one reconfiguration window to the next. If the links are incorrectly reversed for the current traffic pattern then a performance penalty will occur. Therefore, we use ML techniques to improve the accuracy of our LCs.

Machine learning algorithms have been used in network/CMP applications for various reasons such as optimizing wireless sensor topology [34], detection of false memory sharing [35], and management of network power [36]. Various machine learning algorithms are implemented in these designs such as decision trees and artificial neural networks. We use decision trees for predicting the direction of traffic in QORE due to the simplicity during the testing phase. Testing in decision trees is simple because the algorithm uses a few comparisons instead of more complicated operations such as multiplication or addition. Training for the machine learning algorithm can be done offline so that it does not affect the performance of the applications. An example decision tree is shown in Fig. 5. Each node in the tree is a input, or feature, used to predict the target output. The target output of a decision tree is a discrete class. In the case of x links, our three output classes are EAST, WEST, or BOTH and for y links the output classes are NORTH, SOUTH, or BOTH, indicting where the majority of the traffic is moving.

R0 +x Features

- *Link difference* = (Pkts sent on +x) – (Pkts received from +x)
- *Request difference* = (Requests sent on +x) – (Requests received from +x)
- *Response difference* = (Responses sent on +x) – (Responses received from +x)
- *R0 core buffer utilization*
- *R1 core buffer utilization*
- *Core buffer difference* = (R0 core buffer utilization) – (R1 core buffer utilization)
- *R0 +y buffer utilization*
- *R1 +y buffer utilization*
- *R1 +x buffer utilization*
- *Buffer difference = (R1 –x buffer utilization) – (R0 +x buffer utilization)*
- *Other direction difference* = (Pkts sent on others directions for R0) – (Pkts sent on others directions for R1)
- R0 +y link difference
- R1 +y link difference
- R1 +x link difference
- R4 +y link difference
- *R0 core pkt difference* = (Pkts sent by R0 cores on +x) – (Pkts received by R0 cores from +x)
- *R1 core pkt difference* = (Pkts sent by R1 cores on -x) – (Pkts received by R1 cores from –x)

Fig. 6. List of possible features for predicting traffic on the +x links of router 0.

In order to train the decision tree, a set of input features must be engineered. Fig. 6 shows a list of possible features that can be used to detect the traffic on the +x links of router 0 (links a, b, c, and d in Fig. 4). Note that these links can also be labeled as the $-$x links of router 1. The features include various link and buffer utilizations from router 0 and surrounding routers. Initially, the features were selected based on intuition of good predictors. For example, packets using the $-$x links of router 2 can possibly use the $-$x links of router 1 in the future. The algorithm used to build the tree will refine our list of features and use only the features which are most useful in predicting traffic direction. Every R_w cycles the features will be collected, stored in expanded LSTs, and used to predict the outcome of the next R_w cycles. Each LC will be implemented with a decision tree but the features will vary slightly depending on the location of the links in the network.

We use the ID3 algorithm [37] to train the decision trees. The ID3 algorithm, shown in Algorithm 2, uses a set of training data, D , and a set of features, F . At each node in the tree, the algorithm finds one feature, X , which has the largest information gain. The training data D is then partitioned so that all examples that have the same value X_i for feature X are put into a new dataset D_i. The ID3 algorithm is then recursively called on the new data set and on a new set of features that is without feature X . The algorithm recursively builds the rest of the tree with the terminating condition being all examples in D have the same label. However, we modify the terminating condition to limit the size of the tree to three levels so that the number of comparisons during testing is reduced to only three. The implementation of the design tree has a minimal overhead. Results from the Synopsys Design Compiler using a 40 nm technology library show a power (dynamic and leakage) of approximately 3.99 μW, an area of 29.81 $\mu\, m^2$, and a timing of 0.17 ns. In Section 5.7 we show the decision tree and the accuracy of our decision tree in predicting traffic flows.

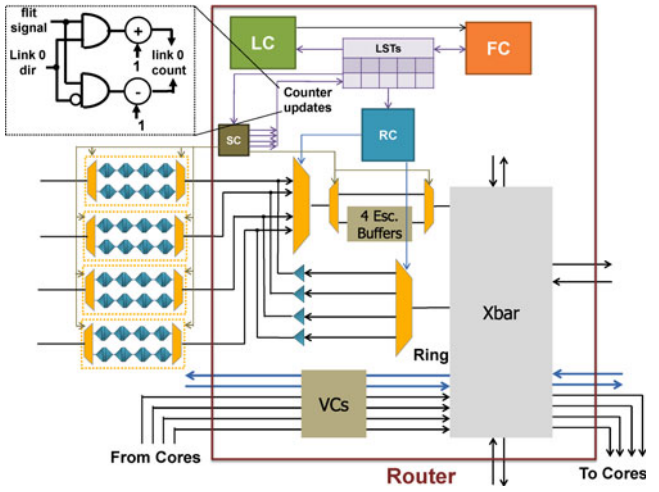


Fig. 7. Router Microarchitecture showing inputs/outputs, LC, FC, and RC.

Algorithm 2. ID3 algorithm used to build the decision tree [37].

ID3(Training data D, Features F):
  if all examples in D have the same label:
    return a leaf node with that label
  let $X \in F$ be the feature with the largest information gain
  let T be a tree root labeled with feature X
  let $D_1, D_2, \ldots, D_k$ be the partition produced by splitting D on feature X
  for each $D_i \in D_1, D_2, \ldots, D_k$:
    let $T_i = \text{ID3}(D_i, F\text{-}\{X\})$
    add T_i as a new branch of T
  return T

4.5 Router Architecture

Fig. 7 shows the router microarchitecture of QORE. The four links to the left of the router can act as outputs or inputs. When acting as an output, the data comes from the crossbar and is demultiplexed onto the four channel buffers. As an input, the data is multiplexed into the crossbar. After a signal is multiplexed, it is normally sent straight to the crossbar. However, it can be sent to an escape buffer. This escape buffer is used to move packets from the channel buffers when the links are reversed. They are also used to avoid deadlocking [38] as explained in Section 4.6. When the escape buffers are full the upstream router will receive a congestion signal and will not send packets to the channel buffers; therefore, guaranteeing that the escape buffers will have enough room to flush out the channel buffers. Four buffers are used because at most four channel buffer lines will reverse direction. Therefore, the four escape buffers ensures that a packet will have a buffer to go to when the links reverse. Each time a flit traverses the links, the counters in the LSTs are incremented or decremented based on the link direction. The inset in Fig. 7 shows the counter for link 0. When a flit traverses a link, it signals the counters and increments the $flitcount$ in the LST if the direction is in or decrements the $flitcount$ if the direction is out. The LC and FC blocks access information from the LSTs as described in the previous section. The route computation (RC) is modified to determine which link to send data on in addition to which direction to send the packet. The link decision is based on which link has the lowest count in the LST. Therefore, the traffic will be spread evenly among the links. The switching control (SC) sends the release (rel) and reverse (rev) signals to the channel buffers. When there is contention at the crossbar or downstream router, the SC notifies the channel buffers to store the data by setting the rel signal to 0 as explained in Section 3. The SC also reads the LSTs to obtain the rev signal, notifying the channel buffers of the correct direction.

4.6 Deadlock Avoidance and Reliability Concerns

In conventional NoCs, XY routing algorithm is used to avoid deadlocks by avoiding turns (Y-to-X). However when links reverse, if not handled properly, there is a potential for deadlock as communication in one direction can be cut off leading to starvation. In QORE, we avoid deadlocks by a) maintaining connectivity, thereby, eliminating starvation, b) using escape VCs to flush out channel buffers during reversing, and c) keeping packets on the backup ring network until their destination is reached. To prove that our network is deadlock-free we examine the three possible states of the N links between routers:

Case I: Zero to N-2 links are faulty. In order to prevent deadlocks, connectivity must remain between routers. FC algorithm 1 first assigns one link to the non-majority direction then assigns the remaining links to the majority direction. This ensures that there is always a connection in both directions. Conventional deadlock-free algorithms such as XY routing can, therefore, be applied and deadlocks are completely avoided.

Case II: N-1 links are faulty. Again, in order to prevent deadlocks, connectivity must remain between routers. However, in this case only one link is available. The algorithm of the FC will assign this one link to the majority direction. Then at 60 percent of R_w, FC will change the $Direction$ field in the LST to the opposite direction. This will cause the link to flush out the data from the channel buffers to the escape VCs located at the downstream router, thereby, allowing packets to be sent in the opposite direction. Therefore, 60 percent of R_w will be allocated to the majority direction and 40 percent of R_w will be allocated for the opposite direction, providing full connectivity.

Case III: All N links are faulty. In this case, no channel buffers are available and protocol states that packets must use the backup ring network to proceed. To avoid deadlocks and livelocks, packets must remain on the backup ring network until their destination is reached. We ensure the packet stays on the ring by adding a one bit $ring$ field to the packet that indicates to the RC whether or not the ring network should be used. When the ring bit is "1", the RC will always send the packet on the ring even if router links are available. If the ring bit is "0" then the router links must be used. To avoid circular dependencies once on the bidirectional ring, a separate set of VCs is allocated to each direction.

Protocol deadlocks can be avoided since each link has two buffer lines. One buffer line can be assigned to requests while the other is used for response traffic. Other than

TABLE 1
Cache and Core Parameters Used for Splash-2, PARSEC, and SPEC2006 Application Suite Simulation

Parameter	Value
L1/L2 coherence	MOESI
L2 cache size/assoc	4 MB/16-way
L2 cache line size	64
L2 access latency (cycles)	4
L1 cache/assoc	64 KB/4-way
L1 cache line size	64
L1 access latency (cycles)	2
Core Frequency (GHz)	5
Threads (core)	2
Issue policy	In-order
Memory Size (GB)	4
Memory Controllers	16
Memory Latency (cycle)	160
Directory latency (cycle)	80

deadlocks and livelocks, another concern may be the issue of the fault tolerant components themselves failing such as the backup ring network or the fault controllers. The backup ring network adds redundancy to links between routers. Moreover, since this backup ring network does not use reversible channel buffers, it has 10 less transistors at every repeater creating a more robust connection between routers. For the LC, FC, and LSTs, since they have a very small overhead, as shown in Section 5.1, these components would be ideal for dual modular redundancy (DMR) or triple modular redundancy (TMR).

5 Evaluation

In this section, we first consider the overhead for our reconfiguration controllers and reversible buffers. Next, we evaluate the fault tolerant performance of QORE compared to the Ariadne [13], Vicis [14] networks by evaluating throughput and power on synthetic traffic as well as speedup on real benchmarks. Lastly, we consider the effect of our reversibility on the overall performance of QORE when no faults are present by comparing to BAR [23] which is not a fault tolerant network.

For open-loop measurement, we varied the network load from 0.1-0.9 of the network capacity. The simulator was warmed up under load without taking measurements until steady state was reached. Then a sample of injected packets were labeled during a measurement interval. The simulation was allowed to run until all the labeled packets reached their destinations. All designs were tested with different synthetic traffic traces such as Uniform Random (UN), non-uniform random (NUR), Bit-Reversal (BR), Butterfly (BFLY), Matrix Transpose (MT), Complement (COMP) and Perfect Shuffle (PS).

For closed-loop measurement, the full execution-driven simulator SIMICS from Wind River [39] with the memory package GEMS [40] was used to extract traffic traces from real applications. The Splash-2, PARSEC, and SPEC CPU200 workloads were used to evaluate the performance of 64-core networks. Table 1 shows the parameters for the cache and core used for the Splash-2, PARSEC, and SPEC2006 benchmarks. We assume a two cycle delay to access the L1 cache, a four cycle delay for the L2 cache, and a 160 cycle delay to access main memory. The power and area results were estimated using the Synopsys Design Compiler with the 40 nm TSMC technology library.

TABLE 2
Power Overhead for the Components of One Router

	Baseline	QORE	Percent Diff.
Storage	111.6 mW	19.8 mW	−82.3%
LC	0	96.27 nW	-
FC	0	96.64 nW	-
Control Block	0	16.32 nW	-
Link	(2×96 bits)	(6×32 bits)	
	307.2 mW	307.2 mW	0%
Crossbar	(8×8)	(9×9)	
	67.4 mW	86.2 mW	+27.9%
Total	486.2 mW	413.2 mW	−15.0%

For fair comparison, every network had four VCs per input and each network was assumed to have a concentration of four cores to a single router as this has been shown to minimize energy and latency while allowing a larger number of cores on a chip [41]. Additionally, we maintained similar bi-sectional bandwidths for each network. The conventional router design (both Ariadne and Vicis) will have two links between each router (one for each direction) and QORE has at most six links between routers (four reversible links, two unidirectional links for the ring) for a ratio of 1:3. However, the bandwidth of each link in QORE is 32 bits/cycle so the total bandwidth between routers will be 192 bits/cycle. Therefore, each link in the conventional design will be 192/2=96 bits/cycle which is 3X the bandwidth of a QORE link. We have assumed that the backup ring network is fault-free and the packet size is four flits each 128 bits.

5.1 Power, Area, and Timing of Reversibility Overhead

Table 2 shows the power overhead for the network components of one router estimated from the Synopsys Design Compiler with a nominal supply voltage of 1.0 V and an operating frequency of 2 GHz. A buffer for the baseline design is a a four flit register buffer and a buffer for QORE is a four stage reversible channel buffer. Each router, in either design, contains 32 buffers (four inputs × eight buffers). The buffers in QORE consume 19.8 mW of power; approximately 82.3 percent less than the baseline register buffers. The amount of leakage power for the reversible channel buffer was found to be 2.44 nW. The overhead of the LC and FC is approximately 96 nW of power and a timing of 0.07 ns. The power of the control block used to switch MFC states was found to be approximately 16.32 nW. The power is a minimal fraction of the total router and the timing is within or clock period. The link power for both baseline and QORE are equal since the total link bandwidth is kept equal. An crossbar power overhead of 27.9 percent is due to the backup ring network in QORE leading to a slightly larger crossbar.

Table 3 shows the area overhead of each router component. The buffers in QORE occupy 147,392 μm^2 which is 3.4× more area than the baseline register buffers. However, unlike register buffers and conventional channel buffers, our channel buffers serve three functions: storage,

TABLE 3
Area Overhead for the Components of One Router

	Baseline (μm^2)	QORE (μm^2)	Percent Diff.
Storage	43,712	147,392	+237.2%
LC	0	1.41	-
FC	0	1.42	-
Control Block	0	83.97	-
Link	(2×96 bits)	(6×32 bits)	
	23,629	23,629	0%
Crossbar	(8×8)	(9×9)	
	580,007	622,418	+7.3%
Total	647,348	793,442	+22.6%

reversibility, and a link repeater. The area overhead of the LC and FC components are approximately 1.4 μm^2 which is minimal compared to the other router components. The area overhead of the MFC control block was found to be 83.97 μm^2. The timing for our reversible channel buffers was estimated to be 0.39 ns which is within our specified clock period of 0.50 ns. The critical path of the four stage reversible channel buffers was composed of eight pass gates (0.22 ns) and four non-reversible channel buffers (0.17 ns). The timing of the critical path as well as estimate of power and area accounted for all additional wiring required between routers.

5.2 Speedup on Real Applications

The speedup of BAR (B), QORE (Q), Ariadne (A) relative to Vicis (V) for different real applications is shown in Fig. 8. The networks were simulated on all applications; however, we only have space to show four applications in the figure. QORE reconfigures its links every $R_w = 50$ cycles. Different values of R_w are evaluated in our prior work [28]. Before runtime, faults were randomly inserted into a percentage of links ranging from 0 to 50 percent. Since BAR is not a fault tolerant network, it is only shown for 0 percent faults. At 0 percent faults, the performance optimized BAR has the largest speedup for all applications as expected. At low to medium faults (0-30 percent), QORE has an average speedup of 1.68× across applications for all benchmarks. At a high number of faults (40-50 percent), QORE has a worse speedup of 0.51× on average. However, this can be misleading because the high number of faults causes Ariadne and Vicis to be split into small subnetworks. Subnetworks are very undesirable because cores from one subnetwork will not be able to communicate with cores from another subnetwork. QORE always maintains connectivity through the backup ring network. The number of subnetworks in Ariadne and Vicis increase with the number of faults from four subnetworks at 30 percent faults to 13 subnetworks at 50 percent faults. Subnetworks partition the chip, blocking communication to many cores. The subnetworks, therefore, lead to a false increase in speedup as also observed in [13]. Whereas, the reversibility of links makes QORE more resilient to communication blocking.

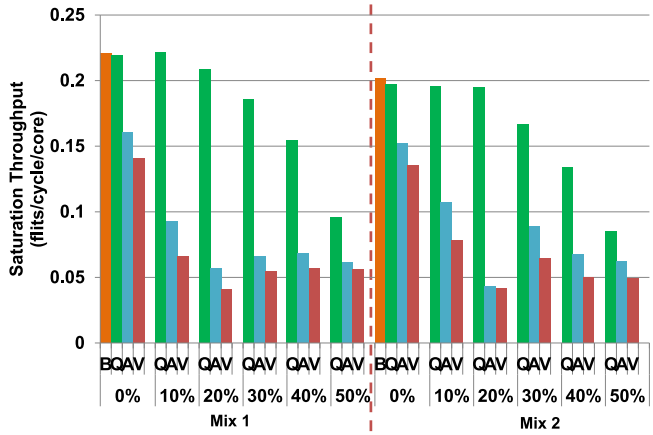


Fig. 9. Saturation throughput for varying percentage of link failures for different traffic mixes for BAR (B), QORE (Q), Ariadne (A) and Vicis (V).

5.3 Network Throughput with Faults

The saturation throughput of the networks for different synthetic traffic mixes is shown in Fig. 9. Four different types of traffic mixes we examine are shown in Table 4 using the abbreviations defined previously in Section 5. However, only two mixes are shown due to space constraints. Each mix randomly cycles through each pattern every $TP = 250$ cycles. QORE reconfigures its links every $R_w = 50$ cycles.

In Fig. 9, QORE consistently has similar throughput to BAR and a higher throughput than both Ariadne and Vicis. Averaged over each traffic mix and fault percentage, QORE's saturation throughput is 2.3× and 2.9× higher than Ariadne and Vicis, respectively. Similar to the speedup results, an increase in throughput can be seen in all mixes when the fault percentage changes from 20 to 30 percent. Again, this is due to link faults causing the network to be

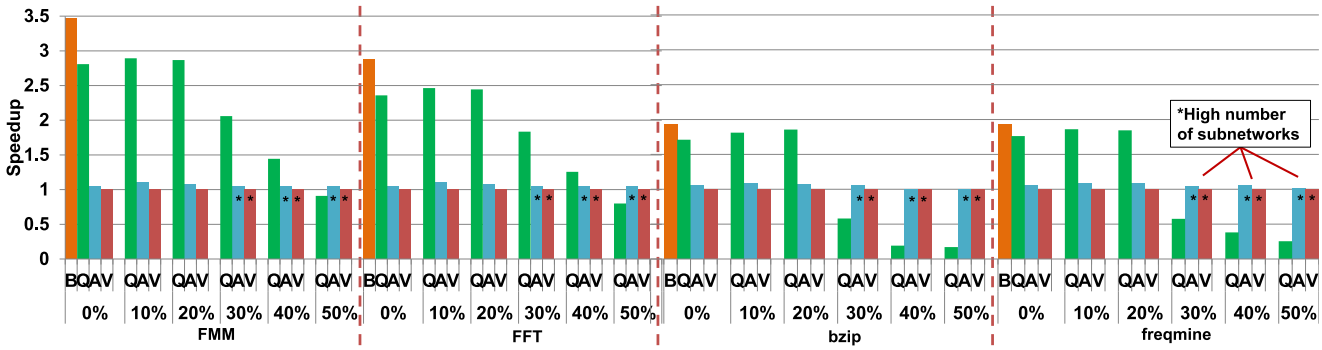


Fig. 8. Speedup of relative to Vicis with varying number of faults where BAR (B), QORE (Q), Ariadne (A) relative to Vicis (V) for 64 cores with SPLASH-2, PARSEC, and SPEC CPU2006 benchmarks.

TABLE 4
Traffic Mixes

Mix	Patterns
Mix 1	BR, BFLY, COMP
Mix 2	NUR, BR, PS
Mix 3	UN, BFLY, MT
Mix 4	UN, BR, COMP, PS

partitioned into smaller subnetworks. When the faults increase to a high percentage (40-50 percent), few flits are sent on a network that has many cores, so the throughput (flits/cycle/core) starts to decrease again.

From 0-20 percent faults, QORE only sees a drop in performance of 3.5 percent averaged over all mixes compared to an approximately 70 percent drop for Ariadne and Vicis. QORE is able to sustain performance due to the adaptability of its links. When a wire between two routers is faulty in Ariadne or Vicis, then all communication between those two routers is blocked even if other wires are non-faulty. With many faults, this limits the number of paths in the network. Therefore, many packets are sharing the same paths which causes a drastic increase in contention for links. QORE, on the other hand, can overcome one or more faulty wires by reversing the available non-faulty links. Reversibility preserves paths between routers which relieves contention. Maintaining minimal contention for links is a main factor for maintaining high throughput.

5.4 Packet Latency with Faults

Fig. 10 shows multiple plots for the packet latency at various fault percentages for traffic mix 1. Latency plots at 10, 20, 40, and 50 percent faults were not shown due to space constraints. At 0 percent faults, BAR saturates at the highest load due to its adaptability, and fine-grained flit transmission. The low load latency for both BAR at 0 percent faults and QORE for all faults is higher than both Ariadne and Vicis. This is due to the serialization delays combined with narrow links in BAR and QORE. However, QORE saturates at a higher load for most fault percentages. At 10, 20, and 30 percent faults, QORE saturates at least 77, 160, and 150 percent higher than Ariadne and Vicis. Faults in Ariadne and Vicis can easily shut down communication between routers. The fault tolerant schemes in these networks forces many packets to take additional hops to reach their destinations because they must move around routers. The increase in hop count greatly increases packet latency for the Ariande and Vicis networks. QORE is able to route more packets minimally to their destination to keep latency low. At 50 percent faults, Ariadne and Vicis saturate 87.5 percent higher than QORE. However, this is due to the many unreachable cores in Ariadne and Vicis which create very small subnetworks resulting in packets with little to no contention.

5.5 Network Power

The total network power for the networks is shown in Fig. 11 for different numbers of link faults and two mixes, although we evaluated the network on all four mixes. For mix 1, QORE saves at least 15 percent power over Ariadne

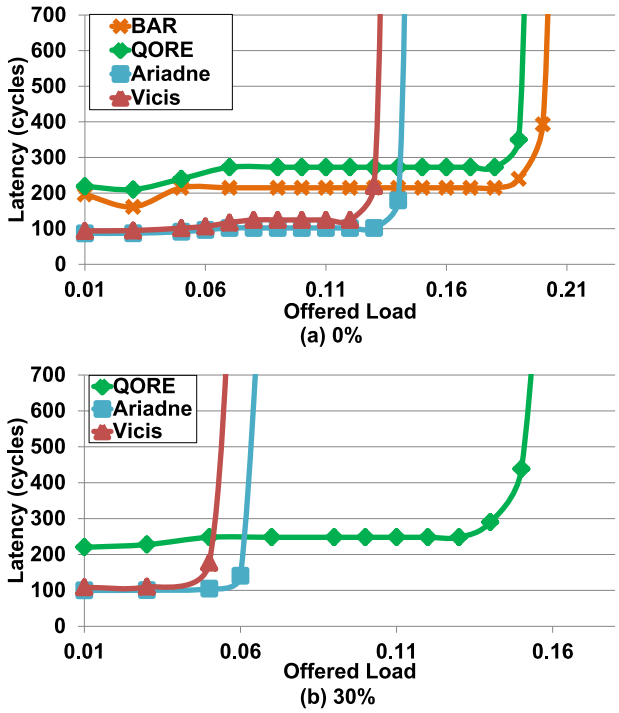


Fig. 10. Latency plots for traffic mix 1.

and Vicis on average. Additionally, QORE saves 25, 22, and 23 percent on traffic mixes 2, 3, and 4. The main contribution to the power savings is the link power. Ariadne and Vicis route around faulty links which many times leads to packets taking non-minimal paths to the destination. QORE can avoid this as long as there is one working link between routers. The only time QORE has the possibility to take a non-minimal path is when the backup ring is used which, in this simulation, only occurred when the fail percentage was 50 percent. As seen in Fig. 11 for 50 percent faults, the power of QORE is higher than Ariadne because of the backup ring routes packets on non-minimal paths for this particular traffic mix. BAR has a power 9.5 percent less than QORE due to the backup ring in QORE which increases the crossbar size by one. However, when the number of faults increases, QORE cannot be compared to BAR since BAR is not a fault tolerant network. Therefore, QORE can save approximately 21 percent power on average while

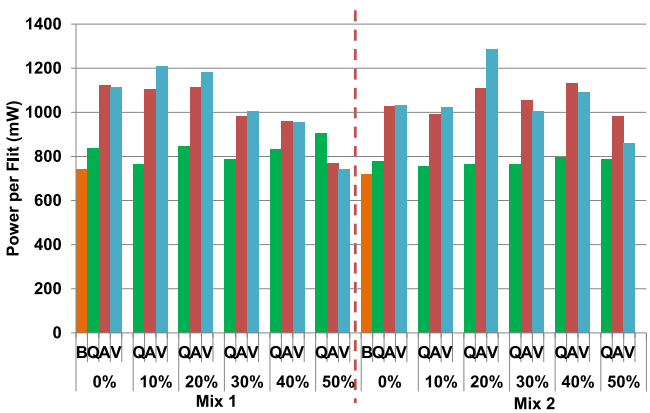


Fig. 11. Network power for different traffic mixes for BAR (B), QORE (Q), Ariadne (A) and Vicis (V).

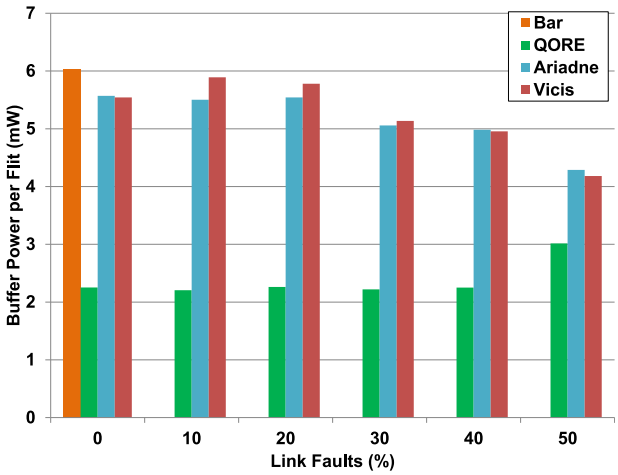


Fig. 12. Buffer power for traffic mix 1.

providing better fault coverage with a speedup 1.3× higher and improved throughput by 2.3×.

To examine the effect of channel buffers on QORE, Fig. 12 shows only the buffer power of a flit. Only traffic mix 1 is shown, other mixes showed very similar results. QORE reduces buffer power by 53.2 percent on average. This savings is due to the low-power channel buffers. The only time register buffers are used in QORE is when a packet is being sent from a core, when a packet is on the backup ring, or when escape buffers are used. The increase in buffer power at 50 percent faults for QORE is due to the increased use of the backup ring.

5.6 Sensitivity Study: Varying Number of Links

In this section, we evaluate the effect of varying the number of links between routers, N. We vary N from two reversible links between routers (Q2) to four links (Q4) to eight links (Q8). As we vary N, the total bandwidth of the links also varies in order to maintain equal bandwidth for a fair comparison. Therefore, Q8 has half the link bandwidth of Q4 which has half the link bandwidth of Q2. Fig. 13 shows the saturation throughput for the various values of N as the link failure percentage increases. Q2 always has the lowest throughput due to the small number of links between routers. Even though Q2 has the highest link bandwidth, the reversibility of the links is very limited. Additionally, the backup ring network is highly utilized in Q2. At a 50 percent link error rate the probability of both links between any two routers failing is 25 percent. On the other hand, at a 50 percent link error rate Q8 only has a 0.3 percent chance of all eight links between any two routers failing. This means that Q8 will have to use the backup ring network much less often. Therefore, at higher fault percentages (approximately greater than 20 percent) Q8 has the highest throughput due to link availability and the rare use of the backup ring. However, since each link in Q8 has half bandwidth of the links in Q4, the saturation throughput of Q4 is higher at lower fault percentages ($\leq$20 percent).

5.7 Accuracy of Decision Trees

In this section, we will show the results of the decision tree used to predict traffic flow. We randomly selected four of the 19 real applications to be used for training (Black-Scholes, Ferret, Bzip, and Radix) and use the remaining 15 applications for testing. Some features have three values: In, Out, or Even. The value is Even if the difference between incoming and outgoing packets is less than or equal to one packet. Other features have two values: Low and High. The threshold to determine Low and High is the median of the feature.

The ID3 algorithm was used to build the trees and the results for the +x links of router 0 are shown in Fig. 14a. The root of this tree is the feature which represents the link utilization on the +x links of router 0 during the previous R_w cycles. This feature alone is the logic used in our baseline LC shown in Algorithm 1. Depending on the link utilization, the tree is further expanded to check other features. For example, another good predictor is the R1 +x link difference because many of the same packets use both the R1 +x links and the R0 +x links. Other links in the network may have trees with different features. For example, the decision tree for the +y links of router 1 are shown in Fig. 14b. The root of this tree is different and represents the R5 +y link difference. The R5 links are directly north of R1 and much of the traffic on the R5 +y links is either coming from or going to the R1 +y links. Buffer utilization is another important feature in both trees because it is likely that packets waiting in buffers will use the links of interest in the future. If all of

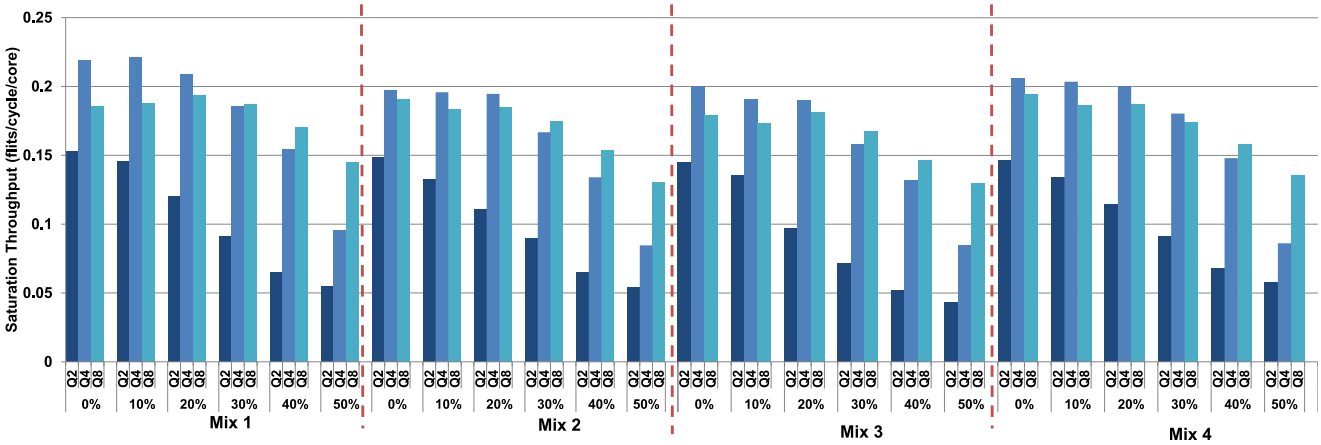


Fig. 13. Saturation throughput for varying percentage of link failures for different traffic mixes for QORE of N values: 2 links (Q2), 4 links (Q4), and 8 links (Q8).

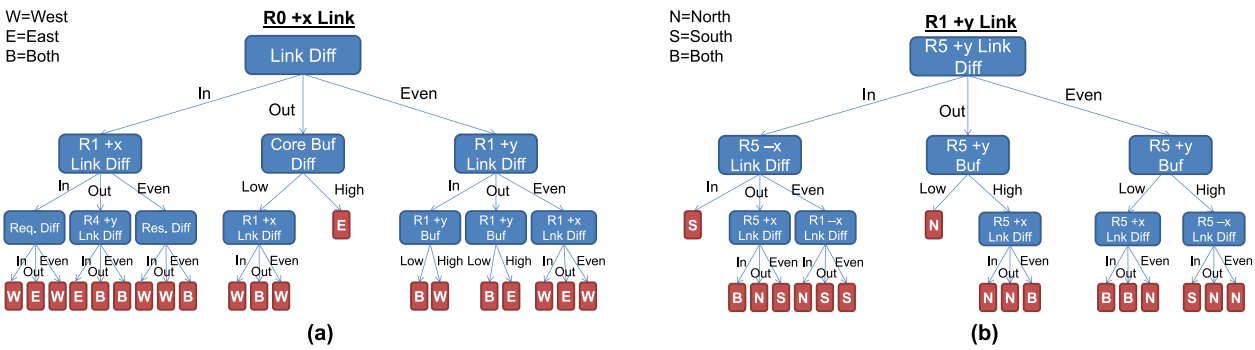


Fig. 14. (a) Decision tree for the +x links of router 0 and (b) decision tree for the +y links of router 1.

the leaves of a node have the same label then that feature is pruned from the tree and replaced with that label. This makes the tree more efficient at test time since only two comparisons are needed to find the label for that branch.

Fig. 15 shows the accuracy of the decision trees for all of the links connected to router 0 and router 1. We compare the decision tree to a uniform random labeling and to one feature—the root of each tree. We can accurately predict the outcome better than a random labeling in every case. On testing data, we can outperform a one feature labeling for the x links and perform equal to or worse for the y links. The y links are more difficult to predict due to the XY routing used in the network. In XY routing, packets use the x links first then the y links. The y links have more possibilities as to which links/buffers were previously used because packets on the y links could have previously been on other y links or other x links. On the other hand, packets on the x links could have only come from other x links. Overall, our results show a 10.35 percent improvement over a random labeling and a 1.25 percent improvement over one feature labeling. The accuracy on training data is also shown in Fig. 15. In the certain cases, such as the R0 +y link, the tree can be trimmed down to one feature to reduce overfitting and improve performance.

6 Related Work

With the increase of soft and hard errors in NoCs due to decreasing technology sizes, much research has gone into the detection and handling of errors. Built-In Self Tests are commonly used to detect errors in systems. Recently, NoCAlert [15] was proposed which detected faults in real-time with 0 percent false negatives. Low overhead checkers were used to detect faults without the need of periodic or triggered-based testing.

As described in the introduction, the Ariadne [13] network uses up*/down* routing to move around faults. Each time a fault was detected, new routing paths were created by transmitting a series of flag broadcasts to all routers. This created a deadlock-free tree network for the irregular topology. The Vicis [14] network also changes its routing algorithm to move around faults when detected. To avoid deadlocks, turn restrictions are placed at certain routers. The Immunet [16] design avoids faults by adaptively routing packets while using escape VCs to avoid deadlocks. Our design differs from these previous works in that we try to avoid additional hops when possible by using reversible links. The implementation of reversibility eliminates the need for routing tables and multiple flag broadcasts to reconfigure the network as seen in Ariadne. Furthermore, in other networks, if one of the two unidirectional links fails then neither link can be used because this would create a one way path to a router. We mitigate this problem by using reversible links as opposed to unidirectional links. Therefore, as long is there is one good link to a router, communication will not be halted.

In [23], a bandwidth-adaptive router was created to increase channel utilization without affecting network latency. BAR increased channel utilization by using narrower channels while also improving performance through adaptive bidirectional channels. Our work also differs from BAR in that we reverse links as well as buffering by using reversible channel buffers. The reversing of buffers as well as links allows the downstream routers to store the increasing number of packets. Additionally, we reverse links/buffers at a coarser granularity to reduce serializer/deserializer overhead. Another reconfigurable design was proposed in BiNoC [22]. BiNoC dynamically reconfigured bidirectional channels to improve performance. We differ from BiNoC in that we reverse buffering as well as links to provide fault tolerance.

	R0 +x Link		R0 +y Link		R1 +x Link		R1 +y Link	
	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data
Random	30.3%	33.4%	31.9%	33.2%	31.6%	34.0%	32.1%	35.1%
One Feature	47.8%	40.1%	44.2%	44.7%	54.0%	44.9%	45.7%	42.4%
DT	56.7%	46.6%	52.4%	40.9%	61.1%	47.2%	53.4%	42.4%

Fig. 15. Accuracy of four different decision trees.

7 Conclusions

With the decreasing technology sizes and increasing number of cores number integrated on a single chip, the design of fault tolerant NoCs that do not degrade performance is critical. In this paper, we propose QORE - a fault tolerant NoC architecture using reversible channel buffers. We use QORE's reversibility for increased performance and to overcome faulty links. We also engineer features and use decision trees to predict traffic direction on the links to improve the link controllers. Experimental results show that a decision tree predicts the direction of the traffic with higher accuracy on average than a predictor based on thresholded link utilization. Our results on real benchmarks (SPEC CPU2006, PARSEC, and SPLASH-2) show an

increase in speedup of 1.3× and improved throughput by 2.3× on synthetic traffic compared to related work. Using the Synopsys design compiler, we show that QORE reduces network power by 21 percent while requiring minimal control overhead.

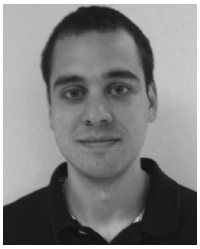
Acknowledgments

This research was supported by the US National Science Foundation (NSF) awards CNS-1318997, ECCS-1342702, CCF-1420681, CCF-1439142, CCF-1054339 (CAREER), ECCS-1129010, CCF-1318981, ECCS-1342657, and CCF-1420718. Dr. Avinash Karanth Kodi is the corresponding author.

References

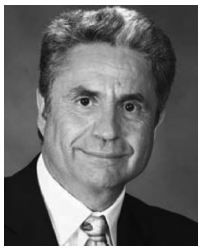
[1] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[2] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Des. Autom. Conf.*, Jun. 18–22, 2001, pp. 684–689.
[3] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep./Oct. 2007.
[4] J. Held, "Single-chip cloud computer: An experimental many-core processor from Intel Labs," presented at the Intel Labs Single-chip Cloud Computer Symposium, Santa Clara, CA, USA, Feb. 12, 2010.
[5] P. Kundu, "On-die interconnects for next generation CMPS," presented at the 2006 Workshop on On- and Off-Chip Interconnection Networks for Multicore Syst., Stanford, CA, USA, Dec. 6–7, 2006.
[6] G. Michelogiannakis, D. Sanchez, W. Dally, and C. Kozyrakis, "Evaluating bufferless flow control for on-chip networks," in *Proc. 4th ACM/IEEE Int. Symp. Netw. Chip*, May 2010, pp. 9–16.
[7] A. K. Kodi, R. Morris, D. DiTomaso, A. Sarathy, and A. Louri, "Co-design of channel buffers and crossbar organizations in NOCs architectures," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2011, pp. 219–226.
[8] G. Michelogiannakis, J. Balfour, and W. J. Dally, "Elastic-buffer flow control for on-chip networks," in *Proc. 15th Int. Symp. High-Perform. Comput. Archit.*, 2009, pp. 151–162.
[9] A. K. Kodi, A. Sarathy, and A. Louri, "Ideal: Inter-router dual-function energy- and area-efficient links for network-on-chip (NOC)," in *Proc. 35th Int. Symp. Comput. Archit.*, Jun. 2008, pp. 241–250.
[10] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, Jun. 2007, pp. 196–207.
[11] M. Hayenga, N. E. Jerger, and M. Lipasti, "SCARAB: A single cycle adaptive routing and bufferless network," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2009, pp. 244–254.
[12] Z. Zhang, Z. Guo, and Y. Yang, "Bufferless routing in optical Gaussian macrochip interconnect," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2685–2700, Nov. 2014.
[13] A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2011, pp. 298–309.
[14] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A highly resilient routing algorithm for fault-tolerant NOCs," in *Proc. Conf. Des., Autom. Test Eur.*, 2009, pp. 21–26.
[15] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "NoCalert: An on-line and real-time fault detection mechanism for network-on-chip architectures," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 60–71.
[16] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," *ACM SIGARCH Comput. Archit. News*, vol. 32, no. 2, p. 198, 2004.
[17] J. Kim, C. Nicopoulos, D. Park, N. Vijaykrishnan, and C. R. Das, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proc. 33rd Annu. Int. Symp. Comput. Archit.*, 2006, pp. 4–15.
[18] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Kilo-NoC: A heterogeneous network-on-chip architecture for scalability and service guarantees," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 401–412.
[19] S. Lin, J. Shi, and H. Chen, "Designing cost-effective network-on-chip by dual-channel access mechanism," *J. Syst. Eng. Electron.*, vol. 22, no. 4, pp. 557 –564, Aug. 2011.
[20] E. Carara, F. Moraes, and N. Calazans, "Router architecture for high-performance NOCs," in *Proc. 20th Annu. Conf. Integr. Circuits Syst. Des.*, 2007, pp. 111–116.
[21] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky, "Bulletproof: A defect-tolerant cmp switch architecture," in *Proc. 12th Int. Symp. High-Perform. Comput. Archit.*, 2006, pp. 5–16.
[22] Y.-C. Lan, S.-H. Lo, Y.-C. Lin, Y.-H. Hu, and S.-J. Chen, "Binoc: A bidirectional NoC architecture with dynamic self-reconfigurable channel," in *Proc. 3rd ACM/IEEE Int. Symp. Netw. Chip*, 2009, pp. 266–275.
[23] R. Hesse, J. Nicholls, and N. Jerger, "Fine-grained bandwidth adaptivity in networks-on-chip using bidirectional channels," in *Proc. 6th IEEE/ACM Int. Symp. Netw. Chip*, May 2012, pp. 132–141.
[24] M. Hayenga and M. Lipasti, "The NoX router," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 36–46.
[25] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, T. Wen, and S. Devadas, "Oblivious routing in on-chip bandwidth-adaptive networks," in *Proc. 18th Int. Conf. Parallel Archit. Compilation Techn.*, 2009, pp. 181–190.
[26] P. Kumar, Y. Pan, J. Kim, G. Memik, and A. Choudhary, "Exploring concentration and channel slicing in on-chip network router," in *Proc. 3rd ACM/IEEE Int. Symp. Netw. Chip*, 2006, pp. 276–285.
[27] S.-J. Chen, Y.-C. Lan, W.-C. Tsai, and Y.-H. Hu, *Reconfigurable Networks-on-Chip*. Berlin, Germany: Springer, 2012.
[28] D. DiTomaso, A. Kodi, and A. Louri, "QORE: A fault tolerant network-on-chip architecture with power-efficient quad-function channel (QFC) buffers," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 320–331.
[29] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "Vichar: A dynamic virtual channel regulator for network-on-chip routers," in *Proc. 39th Annu. Int. Symp. Microarchit.*, Dec. 9–13, 2006, pp. 333–344.
[30] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2008, pp. 72–81.
[31] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The splash-2 program: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Symp. Comput. Archit.*, 1995, pp. 24–36.
[32] L. Chen and T. M. Pinkston, "Nord: Node-router decoupling for effective power-gating of on-chip routers," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 270–281.
[33] J. H. Collet, A. Louri, V. T. Bhat, and P. Poluri, "Robust: A new self-healing fault-tolerant NoC router," in *Proc. 4th Int. Workshop Netw. Chip Archit.*, 2011, pp. 11–16.
[34] Y. Wang, M. Martonosi, and L.-S. Peh, "A supervised learning approach for routing optimizations in wireless sensor networks," in *Proc. 2nd Int. Workshop Multi-hop Ad Hoc Netw.: From Theory Reality*, 2006, pp. 79–86.
[35] S. Jayasena, S. Amarasinghe, A. Abeyweera, G. Amarasinghe, H. De Silva, S. Rathnayake, X. Meng, and Y. Liu, "Detection of false sharing using machine learning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2013, pp. 1–9.
[36] J. Won, X. Chen, P. V. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management," in *Proc. 20th IEEE Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 308–319.
[37] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.
[38] Y. Ho Song and T. M. Pinkston, "A progressive approach to handling message-dependent deadlock in parallel computer systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 3, pp. 259–275, Mar. 2003.
[39] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hllberg, J. Hgberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *IEEE Comput.*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[40] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's genreal execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Comput. Archit. News*, no. 4, pp. 92–99, Nov. 2005.
[41] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proc. 20th ACM Int. Conf. Supercomput.*, Jun. 28–30, 2006, pp. 187–198.



Dominic DiTomaso received the BS and MS degrees in electrical engineering and computer science from Ohio University, Athens, in 2010 and 2012. He is currently working toward the PhD degree in electrical engineering and computer science at Ohio University. His research interests include wireless interconnects, network-on-chips (NoCs), and computer architecture. He is a student member of the IEEE.

Avinash Karanth Kodi received the MS and PhD degrees in electrical and computer engineering from the University of Arizona, Tucson, in 2003 and 2006, respectively. He is currently an associate professor of electrical engineering and computer science at Ohio University, Athens. He received the US National Science Foundation (NSF) CAREER award in 2011. His research interests include computer architecture, optical interconnects, chip multiprocessors (CMPs), and network-on-chips (NoCs). He is a senior member of the IEEE.



Ahmed Louri received the MS and PhD degrees in computer engineering from the University of Southern California, Los Angeles, in 1984 and 1988, respectively. He joined the University of Arizona in 1988 where he is currently a professor of electrical and computer engineering and the director of the High Performance Computing Architectures and Technologies Laboratory. His research interests include computer architecture, parallel processing, interconnection networks, optical interconnects for parallel computing systems, network-on-chips for multi-core architectures and embedded systems. He was a general chair for the 13th Annual Symposium of the High Performance Computer Architecture (HPCA-13), Phoenix, Arizona, 2007. He is a fellow of the IEEE.

Razvan Bunescu received the PhD degree in computer science from the University of Texas at Austin, in 2007, with a thesis on machine learning methods for information extraction. He then joined Ohio University, where he is currently an associate professor of electrical engineering and computer science. His research interests lie in the general area of machine learning, with a focus on applications in computational linguistics, biomedical informatics and more recently computer architecture.