A Reliable Routing Architecture and Algorithm for NoCs

Andrew DeOrio, *Student Member, IEEE*, David Fick, *Student Member, IEEE*, Valeria Bertacco, *Member, IEEE*, Dennis Sylvester, *Fellow, IEEE*, David Blaauw, *Senior Member, IEEE*, Jin Hu, *Student Member, IEEE* and Gregory Chen *Member, IEEE*

Abstract—**Aggressive transistor scaling continues to drive increasingly complex digital designs. The large number of transistors available today enables the development of chip multiprocessors that include many cores on one die communicating through an on-chip interconnect. As the number of cores increases, scalable communication platforms, such as networks-on-chip (NoCs), have become more popular. However, as the sole communication medium, these interconnects are a single point of failure, so that any permanent fault in the NoC can cause the entire system to fail. Compounding the problem, transistors have become increasingly susceptible to wear-out related failures as their critical dimensions shrink. As a result, the on-chip network has become a critically exposed unit that must be protected.**

To this end, we present Vicis, a fault-tolerant architecture and companion routing protocol that is robust to a large number of permanent failures, allowing communication to continue in the face of permanent transistor failures. Vicis makes use of a two-level approach. First, it attempts to work around errors within a router by leveraging reconfigurable architectural components. Second, when faults within a router disable a link's connectivity, or even an entire router, Vicis reroutes around the faulty node or link with a novel, distributed routing algorithm for meshes and tori. Tolerating permanent faults in both the router components and the reliability hardware itself, Vicis enables graceful performance degradation of networks-on-chip.[1]

I. Introduction

Continuously shrinking transistor dimensions enable ever-increasing density on modern microchips: each new technology node facilitates additional cores in chip multi-processors. For example, the Intel SCC [36] contains 48 cores, the Tilera Tile64 has 64 cores [2] and the experimental Intel Polaris chip incorporates 80 cores [59]. However, bus communication and crossbar interconnects have not scaled efficiently: high core counts necessitate efficient, scalable interconnects capable of providing communication among the processor cores. Networks-on-chip alleviate this problem with fast, scalable communication provided by small, distributed, packet-switched routers [11].

Network-on-chip routers communicate via a common interconnect, connecting processor cores, memory controllers, *etc.* At each node (usually a core or memory), a network interface controller (NIC) connects the core to the local router, and converts messages from the core into data packets of varying size for the network. These packets are further divided into flits, the smallest unit of data traveling in the network, which dictates the width of a link connecting two routers. Routers then direct traffic within the network, moving flits from source to destination according to the information encoded in each packet, usually located in the header (first flit) of the packet. In particular, in wormhole routing [40], a single packet's flits may be spread across multiple routers as they traverse the network, until all the constituent flits are collected at the destination. Compared to bus-based systems, network-on-chip designs have the advantage of allowing many messages in flight simultaneously, thus providing efficient communication among many nodes.

While NoCs provide a scalable, distributed communication solution, they are also a single point of failure in a chip multi-processor (CMP). Unlike the cores in a CMP, which are uniform, distributed, and therefore inherently redundant, there is only one communication medium in the chip, constituting a weakness in the presence of faults. Unreliable silicon substrates, brought on by aggressively scaled transistors, threaten the reliability of on-chip communication infrastructures, where a single transistor failure in the NoC could cause the entire chip to fail [38]. The possibility of frequent failures in the field is soon expected to become a reality [5], [55], leading to system failure [6], [18] or even causing security flaws [43].

Transistor failures can be caused by a variety of wear-out mechanisms in highly scaled technology nodes. As transistor dimensions approach the atomic scale, oxide breakdown [56] becomes a concern, since the gate oxide tends to become less effective over time. Moreover, negative bias temperature instability (NBTI) [1] is of special concern in PMOS devices, where increased threshold voltage is observed over time. Additionally, thin wires are susceptible to electromigration [19], because conductor material is gradually worn away during chip operation until an open circuit occurs. Since these mechanisms occur over time, traditional burn-in procedures and manufacturing tests are ineffective in detecting them.

Fault Model. With the reality of decreasing transistor reliability and increasing failures, our goal is to mitigate permanent faults, those that affect the hardware for the remaining life of the chip. Thus, our fault model uses stuck-at failures at the hardware level to portray these permanent faults. Vicis' goal is ensure that all permanent faults in router datapath, and control logic are handled. Furthermore, any hardware additions may be susceptible to the same faults that they aim to address: *faults may occur in the reliability hardware itself.*

Robust, architectural solutions are needed to mitigate the problem of permanent faults. When an error occurs in the field, a typical reliability solution will leverage a *detection* mechanism to identify the problem and notify the system of the failure (Figure 1). Detection can be achieved, for instance, with error correcting codes (ECC) [34] or custom NoC

[1]Earlier versions of this work appeared in [15] and [16]. Included in this paper are the following additional contributions: 1) a study on FIFO unit reliability, 2) extended experimental results on larger networks and with additional workloads, including PARSEC benchmarks, 3) additional results and in-depth presentation of the fault model, and 4) discussion of limitations.

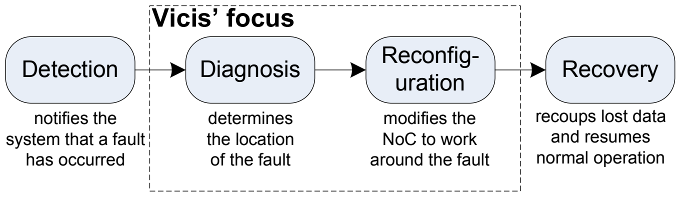
Vicis' focus
Detection
notifies the system that a fault has occurred
Diagnosis
determines the location of the fault
Reconfig-uration
modifies the NoC to work around the fault
Recovery
recoups lost data and resumes normal operation

Fig. 1. **Steps in fault tolerance and Vicis' focus.** Vicis provides a novel diagnosis solution to determine the location of a permanent fault. It then leverages this information to reconfigure the system and overcome the failure.

testing mechanisms [24], [26]. Next, the system undergoes *diagnosis* to determine the fault location. It can then enter a *reconfiguration* phase, isolating the failure or working around it. Finally, *recovery* can take place, recouping lost data with mechanisms such as checkpointing [46], [54]. Following a completed recovery, normal system operation can resume. In this paper, we focus on the diagnosis and reconfiguration phases, which are critical for a high-performance and fault-tolerant system, and we rely on solutions proposed in the literature, such as those cited, for detection and recovery.

Contributions. In this work, we present Vicis, a reliable solution for networks-on-chip with mesh and torus topologies. Vicis leverages a reconfigurable router architecture and routing algorithm. Our solution takes advantage of the redundancy inherent in on-chip networks through a two-level approach. First, it reconfigures individual routers with a novel and flexible NoC router architecture. Second, when errors cannot be contained within a single router, Vicis invokes a novel rerouting solution that modifies the communication paths to bypass the failed node.

As a distributed in-hardware solution, Vicis has the advantage of being able to tolerate many faults, including failures in the reliability components. For systems built on unreliable silicon substrates, Vicis enables graceful performance degradation when transistors inevitably fail.

II. Related Work

Two main approaches for reliable networks-on-chip have been proposed in the literature: one attacking the problem with resilient routing algorithms, and the other with architectural solutions. While many available reliable routing algorithms suggest fault-free communication is a solved problem, many incur significant restrictions, such as limiting the number and location of faults.

Restricted number of faults. Reliable routing algorithms are capable of routing data packets around failures. However, some limit the number of faults that they can tolerate. For instance, an early work in this area is [12]'s reliable router, which can handle a single node or link failure anywhere in the network. [14] can handle $(n-1)$ faults in an n-dimensional mesh and [21] tolerates up to five faults using additional virtual channels. The work in [33] can potentially sustain several faults: the authors provide a backup path around each failed router. As faults accumulate, backup paths form a ring topology. However, the solution fails when additional faults affect the ring network.

Restricted location of faults. Other routing algorithms are able to accommodate more faults, but restrict their location to specific types of "fault regions". A fault region is a subnetwork of a restricted shape that contains faults, and oftentimes correctly functioning nodes must be disabled to satisfy the constraints of the region. The shape of fault regions may be convex [9], [60], or rectangular [58], and sometimes it is also restricted from including the network boundary [57]. Other solutions require fault regions that are polygons [29], **+**, **L** or **T** shapes [7], or contain no holes [17], [48]. Finally, faults may be limited to datapath components, excluding control logic [35], or to links and crossbars [30]

Unrestricted faults. A number of on-chip proposals tackle the problem of unconstrained faults. uLBDR [49] handles routing for any 2D-mesh topology without the need for routing tables. It adds logic to each input port, which facilitate routing around faulty links. However, this approach requires virtual cut-though routing, requiring the entire packet to be buffered at each router. Other solutions address permanent faults by flooding the network to overcome lost network connections, which incurs high performance overhead [45], [51]. Stochastic approaches [4], [53] provide tolerance to permanent and transient faults by means of a probabilistic broadcast mechanism. Immunet [47] routes packets adaptively towards their destinations, based on buffer availability. If necessary, packets switch to a reserved, escape, virtual channel that guarantees that they will reach their destination and avoid faulty links. This channel is aware of the fault locations and routes deterministically in a ring through every node. Upon reconfiguration, a new ring that connects all surviving nodes is formed with a single broadcast, and all in-transit packets drain out via this ring, before updating the routing tables. While the ring guarantees delivery, it dramatically increases latency, since it must remain active during normal operation to ensure deadlock freedom. Additionally, the design requires three routing tables per router, resulting in high area overhead.

Centralized off-chip solutions. Off-chip networks, such as clusters, were the first to address the reliability challenge of unconstrained faults. These resilient routing algorithms can be applied to any irregular topology, and include up*/down* (introduced in Autonet) [52], segment-based routing [37], FX routing [50], L-turn [32], and smart-routing [8]. With these approaches, a central node which runs the reconfiguration algorithm in software. First, the surviving topology is communicated to this central location, which can then use this global knowledge of the functional links to compute new routing tables. Finally, the new routing tables are communicated back to each node. While these centralized reconfiguration algorithms can perform powerful optimizations, communicating the global view of the surviving topology to a central node requires expensive, dedicated hardware. By contrast, on-chip solutions must be designed to meet tight on-chip area budgets.

Reliable router architectures. On chip networks have a tight area and power budget, necessitating simple router structures. Architectural approaches to reliable router architectures include triple modular redundancy (TMR) based approaches, such as the BulletProof router [10]. However, in general, N-modular redundancy (NMR) approaches are expensive, as they

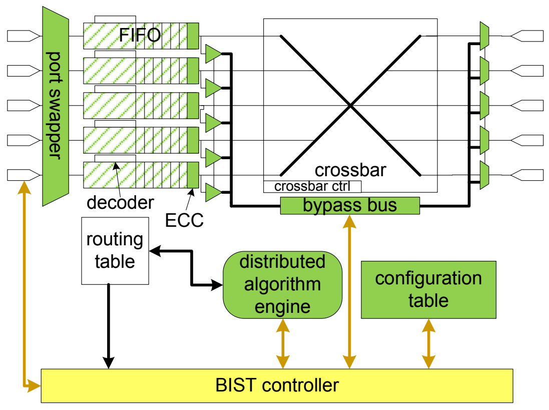

Fig. 2. **Vicis router architecture.** A Vicis-enhanced router includes ECC units, a crossbar bypass bus, a port-swapper, BIST units for diagnosis, a distributed algorithm engine (green/gray) and flexible FIFOs (hashed), in addition to baseline components.

require at least N times the silicon area. Another strategy explores the trade-offs of various levels of redundancy [41]. Other work investigates the reliability of single components, for example a reliability-enhanced crossbar [23]. Reconfiguration is approached by [22] for pipelines, by [31] for link failures and by [28] with modular design. Protection against transient errors has been explored in [42].

III. Router Architecture

Vicis takes a two-part approach to maintain correct execution in a network-on-chip. When a fault is detected, the system goes offline for a Vicis-directed diagnosis and reconfiguration. It first attempts to contain permanent failures within the router, leveraging the inherent structural redundancy in the architecture to work around errors. If the failure cannot be contained within the router, Vicis reconfigures the network around failed nodes and links.

Reconfiguration at the router level is used to contain faults within the router, so that they are not perceivable at the network level as failed links or routers. Figure 2 presents a high-level schematic of a baseline router (in white) with Vicis enhancements (shaded). The baseline router includes input ports and FIFO (first in first out) buffers, decoders, a crossbar, a routing table and output ports. Vicis augments this design with a crossbar-bypass bus to protect against crossbar failures, and with error correcting codes (ECC) to protect datapath elements. Additionally, Vicis can reconfigure the FIFO buffers, the largest router components, to be resilient to a few internal faults. Our port-swapping solution allows Vicis to minimize link failures by reorganizing input ports. Finally, Vicis includes built-in self test (BIST) units to diagnose faulty router components and orchestrate reconfiguration. A complete reconfiguration process requires approximately 152,000 cycles, corresponding to only a few hundred microseconds on a 1 GHz chip. This latency does not appreciably impact application runtime for rarely occurring permanent faults (less than once a day).

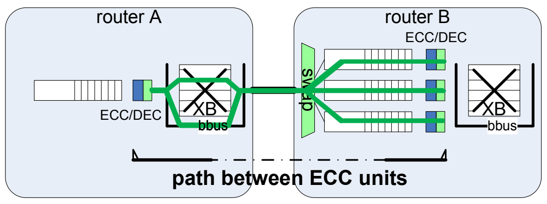

Fig. 3. **Faults mitigated by ECC.** Datapath faults can be corrected by ECC as long as no more than one fault is encountered between two flit-level ECC units. The bypass bus and port swapper provide alternate paths between routers to reduce the number of faults that the ECC units observe. The example in the picture shows six available paths: through crossbar or bypass bus in router A, and through one of three possible FIFOs in router B (the port swapper selects which buffer to use).

A. Crossbar Bypass Bus

In the baseline router, a faulty crossbar would render the entire router inoperable. To address this issue, Vicis adds a crossbar bypass bus, as shown in Figure 2, an alternative path for data that may have to traverse a faulty crossbar path. The crossbar controller is configured to direct traffic to either the crossbar or the bypass bus on a packet basis. If multiple flits simultaneously require the bypass bus, one flit is chosen to proceed first, while the others must wait to use it in subsequent clock cycles. In this manner, the crossbar bypass bus may overcome any number of faults in the crossbar. This spare path provided by the bypass bus allows Vicis to maintain correct operation, even when multiple faults appear in the crossbar. However, in the case of a single fault, the ECC unit is sufficient to overcome the failure.

B. Error Correcting Codes (ECC)

Faults along the datapath can cause data corruption and packet mis-routing. Protecting the datapath with error correcting codes (ECC) enables each component to tolerate a small number of faults while maintaining correct functionality. Previous works have explored the trade-off between energy and reliability by using fine grained error correcting codes [25]. While these studies found that end-to-end ECC was more power efficient than flit-level ECC, they require that packets reach their intended destinations. Errors in header flits that could cause packet mis-routing can only be overcome by flit-level ECC: consequently, Vicis uses flit-level ECC, with an encoder and decoder at the exit of each FIFO. The code adds an additional 6 bits to each 32-bit flit in order to enable 1-bit error correction. Any single fault that manifests along an ECC-guarded datapath section can be corrected when the flit goes through an ECC unit, located in each router at the output of the FIFO buffers. In order to take full advantage of ECC, the BIST unit tracks the location of every datapath fault and, if at all possible, it reconfigures the router to ensure that every distinct path between two ECC units contains at most one fault. If this cannot be accomplished, the router is deemed faulty.

Six paths are possible between two ECC units, depending on the selection of the bypass bus or crossbar (two options) and the configuration of the port swapper (up to three options). Figure 3 illustrates these paths. The port swapper provides three options for the network adapter connection and two options for the other links, but it does not provide all possible swap possibilities. When traversing the network, a flit initially reaches the head of a FIFO in its starting router, goes through an ECC unit for encoding, travels through the crossbar or bypass bus, the link to its next router, the input port swapper and finally reaches its next FIFO. At each unit along the path, faults are diagnosed and cataloged by two BIST units. If two faults accumulate in a same path, the bypass bus and port swapper provide alternative setups to either avoid one of the faults or move one of the them to a different datapath.

For example, consider three faults: one in the crossbar, another in a link, and a third in the default FIFO for the flit in flight. Since the ECC implementation in Vicis can only correct one of these faults, the crossbar bypass bus and input port swapper must mitigate the remaining two. The bypass bus will be used to avoid the crossbar fault, potentially resulting in a loss of performance. The input port swapper will be used to swap in a fault-free input port to the datapath, moving the single-fault input port to another physical link that does not have any other faults. Thus, full functionality is maintained, even with three faults manifesting in the same datapath.

C. Flexible FIFOs

Analysis of a baseline router design informed the selection of our reliable architectural features. Assuming a distribution of faults proportional to the router component area, the largest components — those with the most transistors — are the most susceptible to faults. Thus, we strove to provide additional protection to large components. As shown in Table I, the FIFOs comprise the vast majority of the router, 94% of the baseline router area with 32-flit FIFOs. By comparison, 8-flit FIFOs comprise 80% of the router's area. These results were obtained with a 5-port single-cycle router with a routing table sized for a 3x3 network, synthesized with a 45nm target library. Without any reliability feature, a single fault could cause the entire unit to fail. We thus set out to protect this essential component with a flexible design that can overcome many faults.

FIFOs are comprised of a set of identical registers, and are generally implemented with pointers to determine which register is the head and which is tail. When an item is added to the FIFO, the head pointer is incremented; when an item is removed, the tail pointer is decremented. Thus, the registers are accessed in order, with the first item in being the first item out. The use of identical registers provides an opportunity for a flexible design, with the goal of allowing healthy registers to continue working while skipping faulty ones. Figure 4 shows an example of flexible FIFO operation. The "X" in the figure indicates a FIFO register that experienced a fault. In a baseline FIFO, the head and tail pointers will at some point try to make use of this position, causing the entire FIFO to fail. With this flexible FIFO design, the faulty register can be skipped using pointer redirection, and enabling the FIFO to continue operation with one less register.

To reconfigure the access to registers in a FIFO, the head and tail pointers are indexed through a redirection table mapping sequential FIFO positions to reconfigurable FIFO positions. This allows some positions to be skipped, as illustrated in Figure 5. Effectively, this is similar to incrementing (or decrementing) the head (or tail) counter multiple times before accessing the next functional register, thereby skipping over failed registers. The redirection table is indexed by the head and tail pointers, and provides the index to a functional FIFO entry as output. Additionally, the last entry in the table controls the pointer reset signal, thus allowing the system to adapt to the use of smaller FIFOs as the number of faults increases. With this flexible design, a fault in the FIFO causes only a single register position to fail, maintaining the router's functionality as long as at least one functional register remains.

D. Hard Fault Diagnosis

In order to reconfigure the system, each router must know which of its components contain faults. Furthermore, the use of ECC requires that Vicis knows precisely how many faults are in each part of the datapath. We note that both permanent faults, as well as electrical faults can be diagnosed by Vicis, providing that diagnosis can occur on a faster clock compared

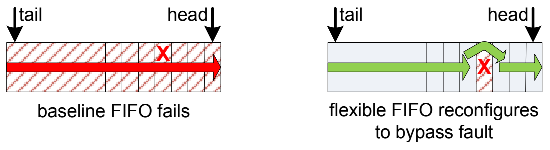

Fig. 4. **Flexible FIFO design.** A flexible FIFO enables Vicis routers to continue operating correctly when using a partially faulty FIFO. While a normal FIFO can fail with a single error, a flexible FIFO reconfigures around the faulty entry.

router component	area (percent)		
	FIFO size		
	8-flit	16-flit	32-flit
crossbar	10.5%	6.0%	3.0%
decoder	3.0%	1.5%	1.0%
FIFO buffers	80.0%	89.0%	94.0%
output logic	3.5%	2.0%	1.0%
routing table	3.0%	1.5%	1.0%
total baseline router	100% $14{,}495\mu m^2$	100% $26{,}173\mu m^2$	100% $49{,}676\mu m^2$

TABLE I
AREA OF THE BASELINE ROUTER BY COMPONENT, FOR DIFFERENT FIFO BUFFER SIZES. THE FIFOS COMPRISE 80-94% OF THE BASELINE ROUTER AREA, AND THUS ARE ESPECIALLY SUSCEPTIBLE TO FAULTS.

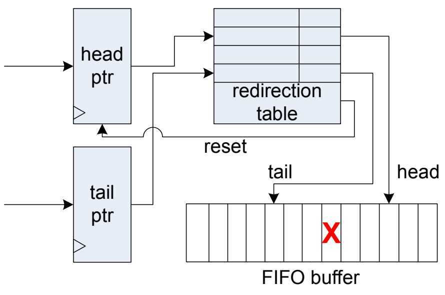

Fig. 5. **Flexible FIFO buffer logic.** The FIFO registers are indexed by the head and tail pointers (counters) through a redirection table, allowing faulty positions to be skipped, and adjusting for fewer FIFO registers.

to normal operation. Control logic is tested with pattern-based testing and datapath faults are counted using datapath testing, as discussed below.

The reconfiguration process begins when one router broadcasts an error status bit through the network, although not necessarily its location, via an extra wire in each link (we assume that a fault detection solution is in place, as discussed in Section I). The initialized BIST unit then performs a distributed synchronization algorithm with other routers' BIST units, ensuring that each BIST in the network runs all remaining routines in lock-step. After synchronization, each component of each router is diagnosed for faults. The diagnosis step does not rely on information from previous diagnostic phases, or from the detection mechanism, thus all permanent faults are diagnosed (or re-diagnosed), regardless of whether they are responsible for triggering this reconfiguration event, or not. Once all components and routers have been tested, faulty components are disabled and normal operation resumes. Since the BIST units are operational only during reconfiguration, they are power-gated off during normal operation for wear-out protection.

Each functional unit is surrounded by wrapper logic, allowing the BIST to assume control during fault diagnosis. The wrapper simply consists of multiplexers for each input, allowing the unit to switch between normal unit inputs and testing inputs from the BIST. A schematic of this structure is shown in Figure 6. Since faults may also manifest in the wrappers themselves, Vicis leverages *interlocked testing* to simultaneously test both the hardware unit, and the wrapper itself. That is, rather than testing the output directly from the module, the BIST unit tests the output after the wrapper mux (as it is indicated by the test flow arrows in Figure 6). This allows the BIST to test both the hardware unit as well as the wrapper logic simultaneously.

The information from a complete BIST run is stored in a configuration table, which contains two bits for each datapath component: crossbar, inter-router link, input port swapper, and FIFO. Each of these units are represented by two bits to indicate fault free, one fault, and two or more faults. Additionally, a redirection table stores flexible FIFO mapping information (described in Section III-C), which is written directly by the BIST. Fault information is later used by the swapping algorithm. The status of non-datapath (*i.e.* control) units is encoded with one bit indicating functional or faulty. This is determined by a signature match or mismatch. In both cases, Vicis is concerned with the fault status of the component, rather than the exact fault location within the component.

1) Datapath Testing: The datapath test determines the number and location of errors in a router's datapath and in the routing table. Units in the datapath need an exact count of faults for each unit so that the maximum number of errors is not exceeded on any path between two ECC units. The test sends patterns consisting of all 1's or all 0's, looking for bit-flip faults. A custom-designed bit-flip count unit determines if the datapath has zero, one, or more bit flips, obviating the need for multiplexers to inspect each bit individually. Each of the 5 FIFO units in a router reuse the same test, limiting BIST unit overhead. Datapath testing requires about 1,000 total cycles.

2) Pattern-Based Testing: Pattern-based tests are used to test the router's control logic. Vicis uses a linear feedback shift register (LFSR) to generate a number of unique patterns, and a multiple input signature register (MISR) to generate a signature. Each unit type tested with pattern-based testing receives the same sequence of patterns from the LFSR, but each has its own distinct signature. Identical units, such as the decoders, have the same signature. A signature mismatch will flag the corresponding unit as broken. Implementation of the pattern-based test is lightweight due to the simplicity of the LFSR and MISR structures. Pattern-based testing requires approximately 150,000 cycles, and runs 25,000 patterns: this is the dominating factor in overall BIST diagnostic runtime.

E. Input Port Swapping

During the initial evaluation of Vicis, we noticed that often a few faults would disable multiple network links or disconnect important processor nodes. To prevent this, we developed *input port swapping* to consolidate several faults into a single link failure, and to provide additional priority for maintaining connected processors. In order to safely route through the network, the routing algorithm (described in Section IV) requires functional bidirectional links. Each link is comprised of two input ports and two output ports, all four of which must be fully functional for the link to be operational. If one of these ports fails, port swapping may be used to maximize functional bidirectional links.

Each input port is comprised of a FIFO buffer and a decode unit, identical for each direction of traffic (see Figure 2). Vicis'

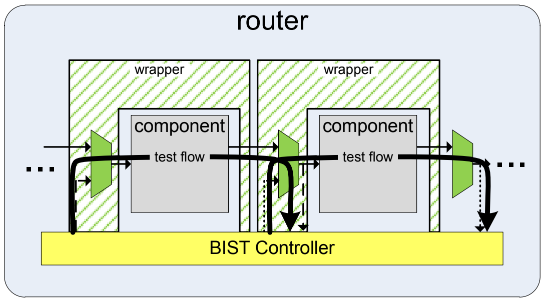

Fig. 6. **Router unit wrappers.** Wrapper muxes allow the BIST controller to access each unit. Testing paths are interlocked through two muxes to enable simultaneous testing of the wrapper and the unit under test.

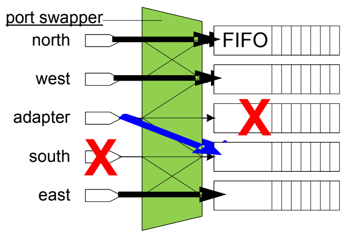

Fig. 7. **Port swapping unit.** The port swapper allows the FIFOs to be connected to different physical links. The local adapter is equipped with more options to maximize the number of cores connected to the faults network.

input port swapper is used to modify which physical links are connected to each input port. For instance, Figure 7 illustrates an example where a fault on the South port and a second fault in the adapter FIFO are consolidated, allowing the adapter link to use the former South FIFO. While it would be possible to include an additional output port swapper at the output ports, their small area and consequent low probability of faults did not warrant the area overhead of an additional swapper. On the other hand, the input ports constitute the majority of the total router area, as discussed in Section III-C and Table I, and therefore are most susceptible to faults. Thus, adding input port swappers provides Vicis with the ability to consolidate the impact of several faults into one, or a few, links.

An example of input port swapper operation is shown in Figure 8. The left side of the figure illustrates five routers in a star configuration: the router in the center has a failed input port and the one on the right has a failed output port. Since these two failed ports are on different links, both links would be considered failed and unusable. However, we note that one of the failed ports is an input port, so the connected physical channel can be changed using input port swapping. The port swapping algorithm reconfigures the system to connect the two failed ports, as shown on the right side of Figure 8. Thus, in this example Vicis takes advantage of the inherent redundancy of the router to increase the number of functional links for the center router from two to three.

In our implementation of the input port swapper, the link to the local network adapter can be connected to three different input ports, while the other links are able to connect to only two possible input ports. The port swapping algorithm is implemented as a greedy algorithm, taking into account the failure status of connected input ports and FIFOs. Additionally, it considers the number of bit failures along the datapath, so as to avoid connecting paths whose bit-errors exceed that which can be corrected by ECC (see Section III-B). Additionally, it prioritizes the local network adapter link, making sure that it is always connected, if at all possible.

Pseudocode for the port-swapping algorithm is shown in Figure 9. The algorithm first eliminates connections that contain control faults and more than one datapath fault. It then connects input ports that have only one viable option. Finally, it selects the highest priority input port among those connected to the FIFO.

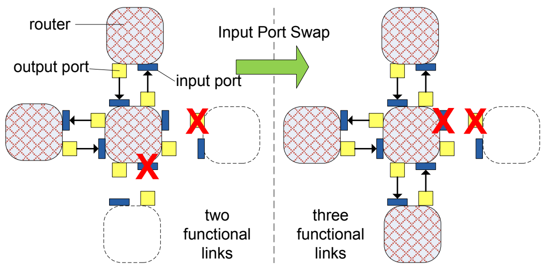

Fig. 8. **Port swapping example.** Routers connected to the network are hashed in the Figure. On the left, an input port failure on the center router and an output port failure affect two different links. By swapping the failed input port to the link connected to the failed output port, Vicis increases the number of functional links from two to three.

```
eliminate connections with control faults
eliminate connections with >1 datapath fault
foreach FIFO:
  selected direction = NULL
  foreach direction connected to this FIFO:
    if direction can only connect to this FIFO:
      selected direction = this direction
      exit loop
if selected direction == NULL
   selected direction = top priority
```

Fig. 9. **Port-swapping algorithm pseudocode.** The port swapping algorithm first disables connections with control faults and too many datapath faults. It then enables connections for which there is only one option. Finally, the algorithm selects the top priority connection among those available.

After the connection is selected, the port swapping algorithm writes the new port configuration to the router configuration table, a set of registers that keeps track of the current link status. This configuration serves to inform the rerouting algorithm as to which links are functional, enabling it to carry the network reconfiguration forward.

IV. Reliable Routing Algorithm

Following reconfiguration within the router, a Vicis-equipped network leverages our reliable and deterministic routing algorithm to work around failed links and failed nodes due to permanent faults. The Vicis routing algorithm reconfigures network routing tables in an offline process based on a basic routing step, which is repeated several times for each network destination. The basic routing step uses local information from neighboring routers to determine connectivity, and applies a set of rules to avoid deadlock (deadlocks occur less than 1 in 10,000 topologies). These rules (disabled turns or links) are selectively relaxed in a preceding checking phase to maximize the connectivity of a faulty network topology. In order to mitigate rare deadlock situations, Vicis can be paired with a deadlock detection mechanism [44]. Once diagnosis is complete, our light-weight routing reconfiguration requires fewer than 1,000 cycles to reconfigure the network. By comparison, Immunet [47] requires approximately 10,000 – 20,000 cycles.

The Vicis routing algorithm is able to tolerate many faults in any link or router, over 1 fault in 2,000 gates in our experiments. Virtual channels are not required, can be used to provide additional performance if available. The algorithm is implemented as a small hardware module included with each router and runs in distributed lock-step.

After router reconfiguration is completed, as discussed in the previous section, transistor-level faults now appear to the routing algorithm as link-level failures. Routers that have become entirely non-functional are represented by nodes with all faulty links. In order to correctly determine routing paths, each router must first discover which of its adjacent links are faulty. Then it works in a distributed fashion with its neighbors to collectively reconfigure their routing tables based on this information. The distributed algorithm uses a basic routing step that follows a set of rules specific to meshes or tori.

```
1:  synchronize_routers()
2:  rules = baseline_rules;
3:  for each router
4:    for each rule
5:      rules = rule_check(rule, rules, router)
6:  for each dest
7:    basic_routing_step(dest, rules)
```

Fig. 10. **Routing algorithm pseudocode.** The algorithm first determines a set of rules that maximizes connectivity while avoiding deadlock using the `rule_check()` function (Figure 11). Once rules have been established for the specific topology, Vicis uses the `basic_routing_step()` (Figure 12) to determine a path to each destination.

```
1: rule_check(rule, rules, router) {
2:   (neighbor1, neighbor2) = elementsOf(rule)
3:   if (not path_exists(neighbor1, neighbor2))
4:     rules -= rule
5:   return rules
6: }
```

Fig. 11. **Pseudocode for rule checking.** Some forbidden turns are enabled during the rule-checking phase in order to maximize network connectivity in partially faulty topologies.

A. Basic Routing Step

Figure 10 shows an overview of the routing algorithm. First, all routers in the network are synchronized (line 1) and a set of baseline rules (disable turns or links) is established (line 2): the algorithm then proceeds in lock step, first determining a set of rules to maximize the connectivity of a faulty network topology (lines 3–5) while avoiding deadlock most of the time. Then, each destination is routed using a basic routing step (lines 6–7) that determines routing paths and uses the rules to avoid deadlock.

Before destinations can be routed, Vicis determines the rules that must be enforced to avoid deadlock. It begins with a set of baseline rules specific to meshes and to tori, and then relaxes some of them to maximize connectivity in a faulty topology. In the Vicis routing algorithm, rules take the form of disabled turns through a single router. Turns are used to avoid deadlock, and are specified as a pair of neighbors $\langle neighbor1, neighbor2\rangle$ where a data packet would execute a routing turn when going from $neighbor1$, through the router under consideration, and then to $neighbor2$. In the case of torus topologies, rules may also be specified as disabled links for the router under analysis. A link is completely specified by the pair of routers that it connects. Rules are intended to avoid deadlock; we have implemented two sets of baseline rules, one specific for 2-D meshes (Section IV-C) and another for tori (IV-D). While in a fully functional topology the complete set of baseline rules is necessary to avoid deadlock, a faulty one may benefit from relaxing some rules to maintain connectivity to partially connected nodes or portions of the network. Vicis addresses this problem by identifying rules that obstruct network connectivity and selectively removing them. Figure 11 shows this procedure, which is applied to each destination in turn: each router executes the `rule_check()` function in order, checking each turn or link. Routes that obstruct network connectivity are those where no path exists between two routers due to a rule (line 3). The connectivity between two routers is determined by the `path_exists()` function, which tests the connectivity between two neighbors of a router. This test is true if the middle router performing the test has a functional link to both of the neighbors in question. If a path does not exist, `rule_check()` removes the offending rule (line 4).

The basic routing step outlined in Figure 12 updates the routing tables in a local router for a specified destination (`dest`). All routers in the network execute the same basic routing step algorithm concurrently and synchronously to route one destination at a time. First, each router performs a **routing table update** (starting on line 2), waiting until routing information for the current destination becomes available (line 3). If the basic routing step is routing to the local router (`SELF`), then the routing table is updated for the current destination (lines 4–5). Otherwise, the router waits to receive a flag from one of its neighbors, indicating that the corresponding neighbor is aware of a route to the destination (line 7). Often, multiple neighbors will offer a route, in which case only one is selected, based on the previously determined rules of the topology and written to the routing table (lines 8–11). When a destination entry is written to the routing table, the corresponding entry is validated, and the direction to that destination is specified. If no route to the destination has been discovered after a timeout, as may be the case in topologies containing many faults, the corresponding routing table entry is invalidated (lines 12–15).

In the **transmission** portion of the basic routing step (Figure 12, lines 17–21), all routers whose destination entry is valid will send a flag to all of their adjacent routers, while other routers are silent. Once a destination entry is routed in one router, that router broadcasts it to its neighbors using a flag, thus allowing the neighbors to discover a path to the destination. Flags are transmitted among routers using the

```
 1: basic_routing_step(dest, rules) {
 2:   // routing table update
 3:   while (dest not routed) {
 4:     if (dest == SELF) {
 5:       rtable_write(dest, SELF)
 6:     } else {
 7:       if (flag_received_from(neighbors))
 8:          neighbor =
 9:          select_neighbor(neighbors, rules)
10:            rtable_write(dest, neighbor)
11:     }
12:     if (timeout) {
13:       rtable_write(dest, INVALID)
14:       break
15:   }
16:
17:   // flag transmission
18:   while (not timeout) {
19:     for each neighbor
20:       transmit_flag(neighbor)
21:   }
22: }
```

Fig. 12. **Basic routing step pseudocode.** The basic routing step is invoked once for each destination, and the algorithm runs on each router concurrently. It updates the local routing table based on connectivity to neighboring routers, communicated through flags, while rules are used to avoid deadlock. Once the local routing table has been written, the router informs its neighbors of the new routing path through flags.

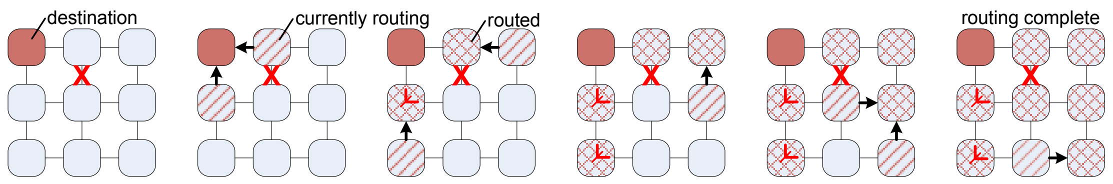

Fig. 13. **Basic routing step example.** In the basic routing step, each router updates its routing table for one destination. Each panel in the figure shows one iteration of the basic routing step when routing to the top left router. The destination router is shaded, nodes being currently routed are hashed and nodes that have completed routing are cross-hatched. Turns are disabled according to the specific topology rules in order to avoid deadlock, and are indicated by**L**-shapes.

physical links connecting them, since normal data traffic is not in flight at this time.

Figure 13 illustrates an example of basic routing step operation for one destination (shaded node) in a 3x3 mesh network with a single fault, indicated by an "X." The destination routes itself (first panel), and the subsequently transmitted flags allow the next two nodes to route to the same destination (cross-hatched nodes in the second panel). In the third panel, two more nodes are routed, but a third node connected via a disabled turn is not routed, because of the baseline rules for mesh topologies. This process continues in the following panels, with more nodes being routed. Routing is completed in the final panel, noting that the final node (hashed) routes to the East since the turn to the West is disabled.

For a network with N routers, the basic routing step must be repeated $N-1$ times for each destination to cover the worst case scenario, where the routers are connected as a long chain due to faults. At the completion of this process, if a router still has an invalid entry for the destination under analysis, then that destination is unreachable from that router.

B. Rules for the Basic Routing Step

During the basic routing step, routers must follow a set of rules to avoid enabling routing paths that could create deadlock loops. The rules consist of a list of disallowed turns or links. A turn through a router $\langle neighbor1, neighbor2\rangle$, can be disallowed by having the router in the elbow of the turn properly configuring its routing table, so that no data packet is routed through the turn. To implement this, during the basic routing step, if the local router has updated its routing table with $rtable\_write(dest, neighbor2)$, then it would not transmit a flag to $neighbor1$.

Rules are enforced at each router, depending on both the topology of the network and the configuration of the faults. For example, a router that knows that a link must be disabled due to faulty hardware will refrain from transmitting flags through that link. In the following discussion, we use the basic routing step to evaluate which rules are necessary to avoid deadlock. Each router will start with a set of baseline rules, removing or adjusting them based on the set of faulty links.

C. 2D-Mesh Routing

A common network topology for large scale chip multiprocessors is the 2D mesh, due to its simple physical implementation. Loops may form naturally when routing in a mesh with faults, so Vicis uses rules to prevent them. In the presence of faults, these rules must be adjusted to maximize connectivity.

In fault-free mesh networks, loops can be avoided by prioritizing turns: each router has four ports, North, South, East and West. When faced with multiple options for routing a single packet, a router prioritizes among these four links in the given order, resulting in deadlock-free routing. This is shown in the first row of Figure 14, where the arrows on the left indicate the traffic patterns and the dashed lines on the right show utilized turns and paths. As shown in the second row of the figure, even a single fault in the network may cause a deadlock loop to form. In this case, the same set of packets are transmitted, but they must use different paths (second row, left). The addition of a turn rule (third row) shows the same set of packets once again, but this time routed deadlock-free through a turn rule in the bottom left corner of the network.

Glass and Ni [20] proposed a technique to prevent deadlock situations by disallowing pairs of turns. One turn must be disallowed for the clockwise direction, and another one for the counter-clockwise direction. The bottom row of Figure 14 shows the application of this technique to avoid deadlock. In our experience, the best results are obtained when disallowed turns are symmetric pairs, for example North$\rightarrow$East and East$\rightarrow$North. This helps in grouping faults, limiting the impact of a single failure. In our solution, we choose to disallow the North$\rightarrow$East and East$\rightarrow$North turns.

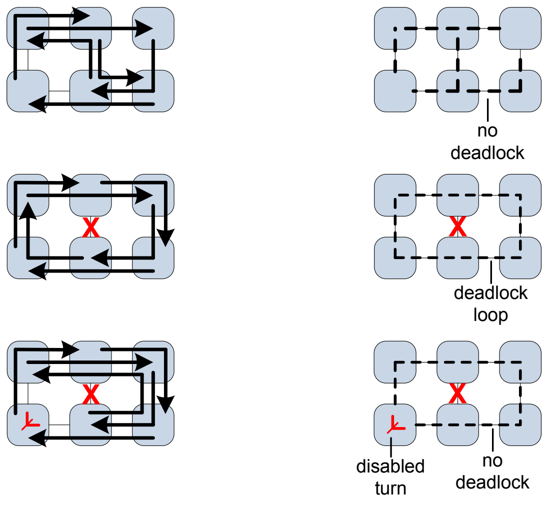

Fig. 14. **Disabled turn example.** The left-hand panel shows routing for the same set of packets under three different conditions: first, a fault-free mesh is routed free of deadlock. The utilized routing paths are shown on the right. The second row shows how a single fault can cause deadlock with the same set of packets, now routed differently due to the fault. Finally, the bottom row shows a turn disabled by Vicis avoids deadlock.

Selectively Removing 2D-Mesh Rules: Strict adherence to the disallowed turn rule may produce an inconsistent network, meaning that when a router can reach another, the corresponding return path could be disabled. For instance, on the left side of Figure 15, a single faulty horizontal link on the North edge of the network prevents six of the routers from obtaining a valid path to the top left router. All of the routers on the West edge of the network can reach this router by simply directing traffic to the North; however, since the East→North turn is disallowed, these routers never transmit a flag toward the East, cutting off the rest of the routers.

In order to contain partitioning in faulty networks, Vicis must identify routers where the turn rules should be re-enabled. This is performed during the rule checking procedure described in Figure 11. We check one turn at a time, so the minimal number of rules are removed in sequence.

D. 2D-Torus Routing

Torus routing presents unique challenges for the basic routing step, and requires a more complex set of baseline rules than meshes.

2D-Torus Rules: Torus networks may form loops around the outside of the network, in addition to the loops that may form in a mesh. Vicis addresses this by leveraging additional rules besides those used for mesh networks. First, it disallows wrap-around links along the top edge of the network, and then it disallows one horizontal link in each row of the network. While the horizontal links prevent a loop from forming in the same row, the vertical link rules along the top edge prevent a zigzagging pattern from looping around the network. Additionally, this prevents loops that would form in the same column. We choose a staggered pattern for the disabled horizontal links in order maintain the performance provided by the torus topology.

Both the vertical link rules and the horizontal link rules need to be checked. First, a horizontal broadcast, where knowledge of broken links propagates horizontally, lifts any horizontal link rules. Since this broadcast starts at the ends of each broken link it is guaranteed to reach every router in the row.

Vertical link rules can be checked in a similar fashion to the turn rules. The router on one end of the link enforces the rule. Vicis applies the basic routing step to the router at the other end of the link and checks if the first router can be reached. If not, the link is needed for connectivity and the rule is removed.

Turn Rule Consistency: A deadlock path in the network can result from removing a turn rule. However, keeping the rule can create an inconsistent network, meaning that when a router can reach another, the corresponding backward path may be disabled and a different return path is instead available. Inconsistent networks may also cause deadlock situations because the outgoing and return paths together may form a cycle. This can be caused by competing paths that traverse around the outside of the network. Vicis resolves this issue by ensuring that paths traversing the outside of the torus network also use the same return path. With this mechanism, Vicis maintains a consistent network and avoids deadlock.

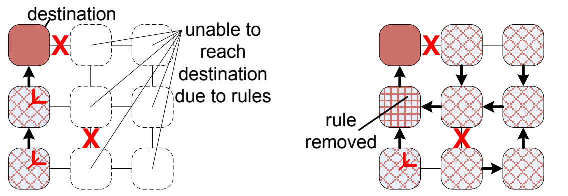

Fig. 15. **Removing rules.** In the presence of faults, disallowed turns can lead to disconnected networks (left). This occurs because the routers on the Western edge have the East→North turn disabled to avoid deadlock. Therefore, we remove the turn rule at the West router, restoring network connectivity (right).

V. Limitations

While Vicis is able to maintain a functional NoC structure by reconfiguring the router architecture and the network topology, there are some limitations in its approach. First, it focuses on diagnosis and reconfiguration, and thus does not provide full system recovery. However, it is amenable to cooperation with a number of recovery mechanisms, such as packet retransmission [39], specialized architectural solutions [13] or checkpointing [54].

Second, while most of the hardware added by Vicis does not affect the critical path, some components do. Specifically, the port swapper and flit-level ECC units are on the critical path of our single-cycle router: from the output of one router's FIFO buffers to the input of its neighbor's. Additionally, while the BIST logic runs offline, the wrappers that allow the BIST to isolate components for testing lie on the critical path between components.

Finally, Vicis' routing algorithm is subject to pathological cases that may prevent deadlock-free routing. These infrequent cases arise in networks with many faults, and are the result of pathological situations in rule checking by the Vicis routing algorithm. The algorithm uses rules, a set of disabled turns or links, to avoid deadlock. These rules are selectively removed to maximize network connectivity in topologies with many faults, as described in Section IV-A. In the majority of situations, this technique enables a greater number of nodes to remain connected to the network. For instance, rare situations arise when faults in a network topology partition the network into disjoint subnetworks connected by a single turn that has been enabled during the rule checking process. Figure 16 shows an example of such a situation. The left panel shows the network before the rule removal: the configuration of faults has created two disjoint subnetworks separated by a disabled turn. Rule checking causes this turn to be enabled, resulting in the topology on the right. The dashed line shows the deadlock loop now formed, passing through the connecting router twice.

To evaluate the impact of pathological cases on various topologies, we examined 4x4, 8x8 and 12x12 meshes and tori with many faults. First, we injected faults at random locations, obtaining one million distinct faulty topologies, and then allowed the network to reconfigure. We then inspected the final routing configuration for deadlocks. Figure 17 shows the results of this study: as shown in the chart, all network configurations exhibit deadlock in less than 1 in 10,000 topologies, when one tenth of the links are faulty. Smaller 4x4 networks were free of deadlock 100% of the time for 2D-meshes, and deadlock for 1 in 10 million cases in 2D-tori,

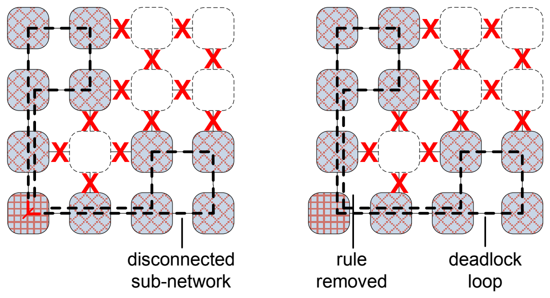

Fig. 16. **Pathological case for large networks with many faults.** A deadlock loop sometimes forms by passing through a router twice.

regardless of the number of faulty links. We noticed that in larger networks, the probability of a routing configuration that allows deadlock increases as the number of faults increases beyond one tenth of the links.

Finally, the design of Vicis leads to a number of interesting possibilities for future work. While Vicis implements a simple parity error correction, other codes are possible. The impact of various ECC encodings on overall system error-tolerance would highlight the trade-offs of performance and correctness. Additionally, some of the techniques applied by Vicis to mitigate permanent faults may also be effective for transient errors, another possible direction for future investigation.

VI. Experimental Results

We evaluated Vicis and the routing algorithm using two models: a slower, but more accurate gate-level hardware description model and a faster architectural model. Simulations were conducted by injecting faults within the system on two different sets of workloads: uniform random traffic and the PARSEC benchmarks [3].

The hardware model was implemented in Verilog HDL, and includes both a 3x3 torus topology and a 3x3 mesh. In both cases, Vicis' reliability enhancements were added to the baseline router. The baseline design is a single cycle, five-port router with one link to a local network adapter, and four links to its neighboring routers. Each router's input is connected to a 32-flit FIFO, which passes through 32-bit data flits. The router was synthesized, and automatically placed and routed to

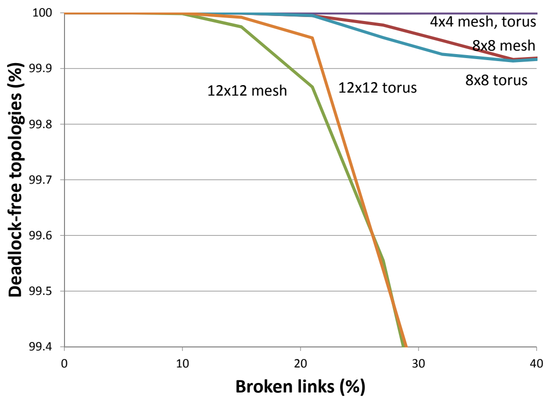

Fig. 17. **Deadlock-free topologies with increasing faults.** All topologies were found to be free of deadlock at least 99.99% of the time with up to 10% of the links broken.

obtain our final simulated netlist. This highly-accurate model was used for simulations, as well to inform the fault model to be used for the architectural simulations.

The architectural model was a custom, cycle-accurate simulator written in C++ with a configuration similar to that of the hardware model. The fast architectural model made it possible to evaluate larger topologies, including an 8x8 mesh and an 8x8 torus. This model also enabled higher testing scalability, making it possible to run longer random tests, and enabling the system to handle the PARSEC benchmarks. The FIFO buffers in the architectural model accommodated 16 flits. Additionally, a statistical model to generate faulty network topologies (described in Section VI-A) was generated, informed by the simulation results of the hardware models.

Test packets were generated at each network-adapter by a random traffic generator which injected traffic to and from all network locations with uniform random probability. Packet length varied from 1 to 10 flits with a uniform random distribution. Additionally, to evaluate correctness, packets injected at each network adapter were checked for arrival at the correct destination with the correct data.

A. Fault Model

Our fault model was designed to reflect the accumulation of permanent transistor faults that occur during the lifetime of a chip. We generated a fault model for our architectural simulator by evaluating the impact of faults at the gate-level and projecting it to the architectural level as link failures. To this end, we injected stuck-at faults at the gate outputs of our reliability-enhanced hardware model in randomly selected locations. Faults were injected in both baseline router components, as well as the additional reliability components. The BIST was an exception, since it can be power-gated during normal operation, and thus is much less susceptible to permanent faults. The hardware model was synthesized, placed and routed prior to fault injection. The random selection of faulty gates was weighted by gate area. This is consistent with the breakdown patterns observed experimentally by Keane, *et al.* in [27]. While this model works well for gate-level analysis, it must be abstracted for high-level architectural evaluations.

Our architectural simulator must be informed of the location of faulty links. Thus, we must map gate-level faults to link-level faults. To this end, we leveraged the fault impact analysis in our gate-level model to form a probabilistic link fault model. We first ran simulations on the HDL design of our Vicis router after injecting faults, and allowed the hardware to reconfigure. With 100,000 distinct RTL simulation results, we built a model mapping gate-level errors to link-level errors. Figure 18 shows the distribution of faults, mapping gate-level router faults to link failures on our 5-port router. Links failures could range from 0, indicating no faults, to 5, indicating that all links were faulty. For example, with 8 faults in a router, all links will be broken with probability 0.2; 1 link will be functional with probability 0.25, etc. This model was used in the architectural simulations to enable fast simulation with industrial benchmarks and longer traces.

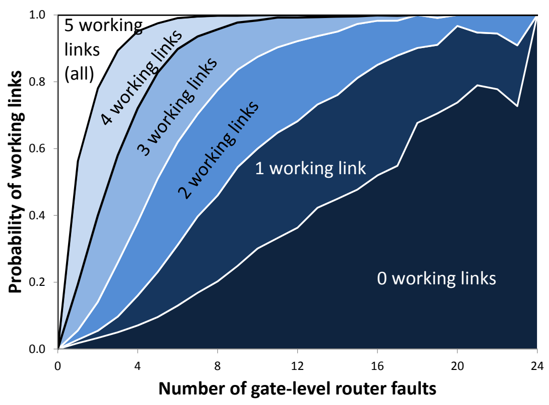

Fig. 18. **Fault model.** The graph shows the distribution of faults among the five router links as a function of gate-level faults. Vicis leveraged 100,000 low-level HDL simulations to form a statistical fault model used by our high-level architectural simulator. The figure shows the mapping of gate-level router faults in the low-level simulations to link failures used in high-level simulation.

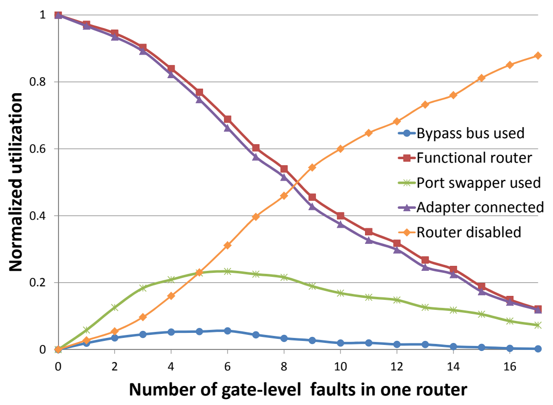

Fig. 19. **Utilization of reliability features with increasing router faults.** The plot reports the probability of a functional (as well as of a disconnected) router over an increasing number of faults and whether the local network adapter is still functional. Additionally, we indicate which reliability features were used to enable a router to remain functional.

B. Fault Tolerance

We first compared the fault tolerance of a network comprised of Vicis routers to a comparable network implementing triple modular redundancy (TMR). TMR provides probabilistic reliability: since the voter takes the most common signal of the three replicated units, it is possible for only two faults to cause the system to fail. In the worst case, a single fault could cause system failure if it occurred in a clock tree or another non-replicable cell. Unlike BulletProof [10] and other prior work that relies on maintaining total functionality, Vicis is able to tolerate many simultaneous faults, including ones that render entire routers useless, since it is able to route around them. Thus, Vicis can maintain near 100% reliability even for a high number of faults, trading off performance for correctness. A key difference between TMR and Vicis is performance. TMR maintains constant, 100% performance until any component loses one of its redundant versions, after which the entire system fails. On the other hand, Vicis enables gracefully degrading performance as faults accumulate.

In our next study on the gate-level model, we tested again a gate-level 3x3 torus network, considering eleven different situations with varying simultaneous faults: 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. The case of 100 faults corresponds to approximately one fault for every 2,000 gates. For each number of simultaneous faults, we considered 1,200 different random faulty topologies. Then, we analyzed each router in the topology, considered how many fault it sustained, whether it was still functional and had a functional local network adapter, and what reliability features it was utilizing. The results are shown in Figure 19. We note from Figure 19 that the input port swapper is very successful at keeping cores connected to the network. As reported in the figure, only a very small fraction of the functioning routers do not have a functional local adapter, as indicated by the closeness of the two curves. The swapper had a high utilization, being used nearly 24% of the time for routers with seven faults.

When considering the utilization of the bypass bus, it was much less often invoked. At seven faults, the crossbar bypass bus was used less than 6% of the time. Two reasons

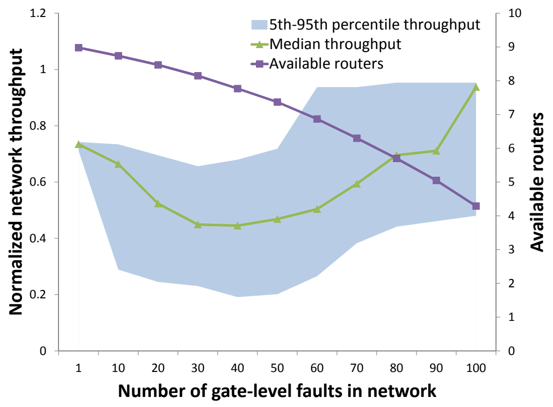

Fig. 20. **Network performance as faults increase** with a 3x3 torus network. Normalized network throughput is shown as the number of faults increases. Throughput is normalized to the bandwidth of the available network adapter links. The shaded region show the 5th-95th percentiles, while the line represents the median.

contributed to this: first, the crossbar is relatively small – less than five percent of the total area of the router and thus suffered a smaller incidence of failures. Secondly, the crossbar is protected by both the input port swapper and the ECC mechanism.

C. Performance in the Presence of Faults

We examined the effect of faults on network performance. Figure 20 shows that network performance gracefully degrades as the number of faults in a gate-level 3x3 torus network increases. The black line (marked with squares) shows the number of connected cores. At 90 faults, with more than 10 faults per router on average, we found that over 50% of the system's cores were still available. The figure also shows the normalized network throughput (marked with triangles). For the first 30-40 faults, median network throughput decreases due to link failures, forcing packets to take longer paths. Beyond 40 faults however, performance begins to increase as a result of the smaller networks formed due to partitioning. This is due to routers becoming disconnected, decreasing the size of the remaining network. Shading indicates the 5th-95th percentile range of normalized throughput.

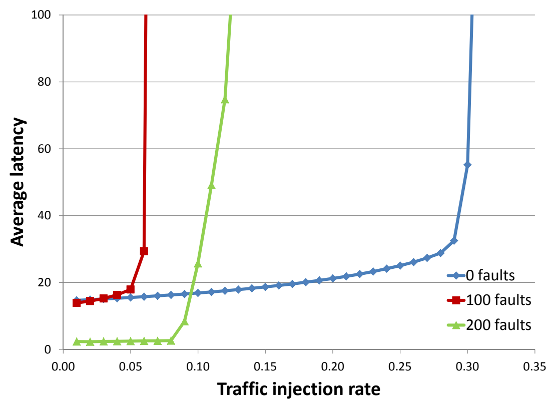

Fig. 21. **Packet latency** as traffic density increases in an 8x8 torus network. The chart reports results for 0, 100 and 200 faults.

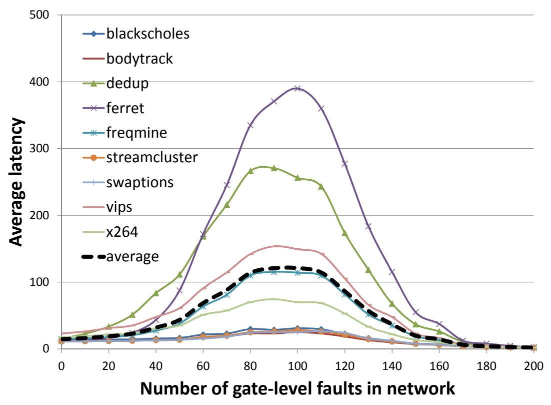

Fig. 22. **Packet latency with PARSEC benchmarks** in an 8x8 torus network. The average is shown by the heavy dotted line, showing network latency increasing as faults increase up to 100. Beyond 100 faults, latency decreases due to decreasing functional network size.

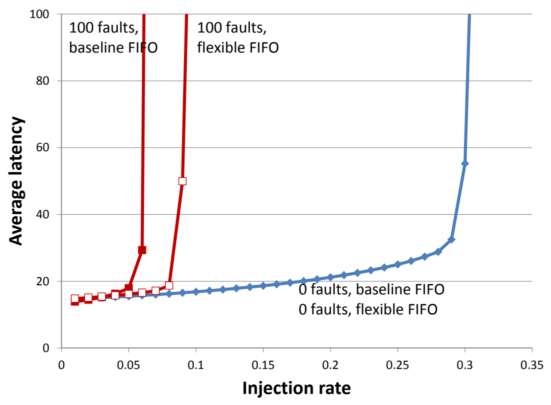

Fig. 23. **Impact of flexible FIFO design** on Vicis' performance in an 8x8 torus network. The plot indicates that the use of flexible FIFOs allows a Vicis-equipped network to hit the latency wall at higher injection rates.

We then examined network latency under different traffic densities using uniform random traffic. For this study, we used the C++ network model and measured the average latency of packets traversing an 8x8 2D-torus network as the density of randomly generated traffic increased. In Figure 21, density is reported as a fraction of the total injection bandwidth of the system, which is fixed and proportional to the number of routers in the network. For low traffic densities, the latency remains under 20 cycles, however, as the density increases, network saturation occurs, resulting in a latency wall. When subjected to faults, the latency wall is reached at lower traffic densities, as indicated in the graph. We noted, however, that the latency wall for 200 faults is actually farther out than that of 100 faults. This is due to a significantly smaller network size at 200 faults due to network partitioning.

For evaluation of Vicis' performance with more realistic workloads, we also evaluated our solution with the set of PARSEC benchmarks [3]. In this experiment (Figure 22), we configured our C++ simulator to an 8x8 torus network and mapped gate-level faults to network link faults as described in Section VI-A. Each data point represents 1,000 random fault-injected topologies with different random seeds. Traffic was injected using traces obtained from an architectural simulation: 100,000 packets were injected. We noticed that similar trends appear as in our previous experiment with random traffic: latency increases as faults increase, up to 100 faults. Beyond 100 faults, the latency begins to decrease, as the remaining functional portion of the network shrinks. The resulting smaller effective network is due to the increasing number of routers completely disabled by faults.

Finally, we investigated the effectiveness of flexible FIFOs. In these experiments, we simulated an 8x8 torus network with the C++ model. A portion of the failures proportional to the area of the FIFOs were injected directly into the FIFOs, resulting in effectively smaller FIFO buffers after reconfiguration. We then ran simulations injecting 100,000 packets of uniform random traffic. 1,000 different faulty topologies were used for each datapoint. Figure 23 shows the average latency curves for the faulty topologies with 100 injected faults. The no-fault case is also shown for reference. First, the chart shows that with no injected faults, the flexible FIFO provides exactly the same performance as the baseline FIFO. As we did earlier, we again note that the fault-injected topology reaches the latency wall sooner. However, with flexible FIFOs, this effect can be mitigated, reaching the latency wall at an injection rate of approximately 0.1, compared to a previous 0.07.

D. Area Overhead

The physical design of the Vicis router was carried out with an automated place and route tool chain after synthesizing with Synopsys Design Compiler, targeting a 45nm technology. The baseline router was also designed in this fashion. The resulting Vicis reliable router with 32-flit FIFOs comprised 74,805μm^2, for an area overhead of 51% compared to the baseline router. This includes both the hardware to implement the routing algorithm, as well as all the architectural features. Table II shows the overhead of each component in the Vicis router in a 3x3 network. Among the reliability components, the BIST logic is the largest.

VII. Conclusions

We have presented Vicis, a reliable network-on-chip that is able to tolerate many faults in both the router components as well as the reliability components themselves. It maintains correct operation in the face of faults, trading off performance for correctness. As the number of failures increases, Vicis

router component	Vicis area (μm^2)	baseline area (μm^2)
crossbar	2,657	1,487
decoder	1,350	395
flexible FIFO buffers (32-flit)	54,303	46,706
output logic	925	644
routing table	585	416
misc	854	28
BIST and reconfig. logic	5,082	-
bypass bus	201	-
ECC	1,544	-
port swapper	603	-
total	**74,805**μm^2	**49,676**μm^2
overhead	**51%**	

TABLE II
AREA BREAKDOWN OF VICIS ROUTER BY COMPONENT.

mitigates errors by reconfiguring both the router architecture and network routing protocol. By leveraging the redundancy inherent in networks-on-chip, and NoC routers, Vicis can maintain high reliability, while incurring a 51% overhead.

A built-in self test at each router diagnoses the number and locations of hard faults. Architecture features including ECC, a crossbar bypass bus and port swapping are then deployed to work around faults. Finally, routers work together to run a distributed in-hardware network reconfiguration algorithm, thus bypassing broken links and routers. We show that Vicis is able to provide significant area and reliability advantages over TMR, tolerating fault rates of over 1 in 2,000 gates.

ACKNOWLEDGMENTS

This work was developed with partial support from the Gigascale Systems Research Center.

REFERENCES

[1] M. Alam. A critical examination of the mechanics of dynamic NBTI for PMOSFETs. In *Proc. IDEM*, 2003.

[2] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 processor: A 64-core SoC with mesh interconnect. In *Proc. ISSCC*, 2008.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proc. PACT*, 2008.

[4] P. Bogdan, T. Dumitras, and R. Marculescu. Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. *VLSI Design*, 2007(95348), 2007.

[5] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *Proc. MICRO*, 2004.

[6] S. Borkar, N. P. Jouppi, and P. Stenstrom. Microprocessors in the era of terascale integration. In *Proc. DATE*, 2007.

[7] S. Chalasani and R. Boppana. Communication in multicomputers with nonconvex faults. *IEEE Trans. Computers*, 46(5), 1997.

[8] L. Cherkasova, V. Kotov, and T. Rokicki. Fibre channel fabrics: Evaluation and design. In *International Conference on System Sciences*, 1995.

[9] A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proc. ISCA*, 1992.

[10] S. Constantinides, K.and Plaza, J. Blome, V. Zhang, B.and Bertacco, S. Mahlke, T. Austin, and M. Orshansky. Bulletproof: a defect-tolerant cmp switch architecture. In *Proc. HPCA*, 2006.

[11] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. DAC*, 2001.

[12] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proc. PCRCW*, 1994.

[13] A. DeOrio, K. Aisopos, V. Bertacco, and L.-S. Peh. DRAIN: Distributed recovery architecture for inaccessible nodes in multi-core chips. In *Proc. DAC*, 2011.

[14] J. Duato. A theory of fault-tolerant routing in wormhole networks. *IEEE Trans. Parallel and Distributed Systems*, 8(8), 1997.

[15] D. Fick, A. DeOrio, V. Bertacco, D. Sylvester, and D. Blaauw. A highly resilient routing algorithm for fault-tolerant NoCs. In *Proc. DATE*, 2009.

[16] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester. Vicis: a reliable network for unreliable silicon. In *Proc. DAC*, 2009.

[17] J. Flich and J. Duato. Logic-based distributed routing for nocs. *Computer Architecture Letters*, 7(1), 2008.

[18] S. Furber. Living with failure: Lessons from nature? In *Proc. ETS*, 2006.

[19] P. B. Ghate. Electromigration-induced failures in VLSI interconnects. In *Proc. Reliability Physics Symposium*, 1982.

[20] C. Glass and L. Ni. The turn model for adaptive routing. In *Proc. ISCA*, 1992.

[21] M. E. Gomez, J. Duato, J. Flich, P. Lopez, A. Robles, N. A. Nordbotten, O. Lysne, and T. Skeie. An efficient fault-tolerant routing methodology for meshes and tori. *IEEE Computer Architecture Letters*, 3(1), 2004.

[22] S. Gupta, S. Feng, J. Blome, and S. Mahlke. StageNet: A reconfigurable CMP fabric for resilient systems. In *Reconfigurable and Adaptive Architecture Workshop*, 2007.

[23] R. He and J. Delgado-Frias. Fault tolerant interleaved switching fabrics for scalable high-performance routers. *IEEE Trans. Parallel and Distributed Systems*, 18(12), 2007.

[24] M. Hosseinabady, A. Banaiyan, M. N. Bojnordi, and Z. Navabi. A concurrent testing method for NoC switches. In *Proc. DATE*, 2006.

[25] A. Jantsch, R. Lauter, and A. Vitkowski. Power analysis of link level and end-to-end data protection in networks on chip. In *Proc. ISCAS*, 2005.

[26] N. Karimi, A. Alaghi, M. Sedghi, and Z. Navabi. Online network-on-chip switch fault detection and diagnosis using functional switch faults. *Journal of Universal Computer Science*, 14(22), 2008.

[27] J. Keane, S. Venkatraman, P. Butzen, and C. H. Kim. An array-based test circuit for fully automated gate dielectric breakdown characterization. In *Proc. CICC*, 2008.

[28] J. Kim, C. Nicopoulos, and D. Park. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. *ACM SIGARCH Computer Architecture News*, 34(2), 2006.

[29] S.-P. Kim and T. Han. Fault-tolerant wormhole routing in mesh with overlapped solid fault regions. *Parallel Computing*, 23(13), 1997.

[30] A. Kohler and M. Radetzki. Fault-tolerant architecture and deflection routing for degradable noc switches. In *Proc. NoCs*, 2009.

[31] A. Kohler, G. Schley, and M. Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 29(6), 2010.

[32] M. Koibuchi, A. Funahashi, A. Jouraku, and H. Amano. L-turn routing: An adaptive routing in irregular networks. In *International Conference on Parallel Processing*, 2001.

[33] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. *Proc. NoCs*, 2008.

[34] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983.

[35] C. Liu, L. Zhang, Y. Han, and X. Li. A resilient on-chip router design through data path salvaging. In *Proc. ASPDAC*, 2011.

[36] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, N. Vangal, Sriram Borkar, G. Ruhl, and S. Dighe. The 48-core SCC processor: the programmer's view. In *Proc. SC*, 2010.

[37] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie. Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori. In *Proc. IPDPS*, 2006.

[38] G. D. Micheli. Reliable communication in systems on chips. In *Proc. DAC*, 2004.

[39] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. D. Micheli. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test*, 22(5), 2005.

[40] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 6(2), 1993.

[41] S.-J. Pan and K.-T. Cheng. A framework for system reliability analysis considering both system error tolerance and component test quality. In *Proc. DATE*, 2007.

[42] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das. Exploring fault-tolerant network-on-chip architectures. In *Proc. DSN*, 2006.

[43] A. Pellegrini, V. Bertacco, and T. Austin. Fault-based attack to RSA authentication. In *Proc. DATE*, 2010.

[44] S. Peng and R. Manohar. Self-healing asynchronous arrays. *Proc. ASYNC*, 0, 2006.

[45] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on*, 2004.

[46] M. Prvulovic, Z. Zhang, and J. Torrellas. ReVive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *Proc. ISCA*, 2002.

[47] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A cheap and robust fault-tolerant packet routing mechanism. *ACM SIGARCH Computer Architecture News*, 32(2), 2004.

[48] S. Rodrigo, J. Flich, J. Duato, and M. Hummel. Efficient unicast and multicast support for CMPs. In *Proc. MICRO*, 2008.

[49] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Proc. NoCs*, 2010.

[50] J. C. Sancho, A. Robles, and J. Duato. A flexible routing scheme for networks of workstations. In *Proc. HPCS*, 2000.

[51] A. Sanusi and M. Bayoumi. Smart-flooding: A novel scheme for fault-tolerant NoCs. In *Proc. SOCC*, 2009.

[52] 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 Journal on Selected Areas in Communication*, 9(8), 1991.

[53] W. Song, D. Edwards, J. Nunez-Yanez, and S. Dasgupta. Adaptive stochastic routing in fault-tolerant on-chip networks. In *Proc. NoCs*, 2009.

[54] D. Sorin, M. Martin, M. Hill, and D. Wood. SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *Proc. ISCA*, 2002.

[55] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. DSN*, 2004.

[56] J. H. Stathis, B. P. Linder, R. Rodrguez, and S. Lombardo. Reliability of ultra-thin oxides in CMOS circuits. *Microelectronics Reliability*, 43(9-11), 2003.

[57] P.-H. Sui and S.-D. Wang. Fault-tolerant wormhole routing algorithm for mesh networks. *IEEE Computers and Digital Techniques*, 147(1), 2000.

[58] M.-J. Tsai. Fault-tolerant routing in wormhole meshes. *Journal of Interconnection Networks*, 4(4), 2003.

[59] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1), 2008.

[60] Z. Zhang, A. Greiner, and S. Taktak. A reconfigurable routing algorithm for fault-tolerant 2D-mesh network-on-chip. In *Proc. DAC*, 2008.

Andrew DeOrio (S'07) is a Ph.D. student at the University of Michigan in the Advanced Computer Architecture Lab (ACAL). He received his B.S.E. and M.S.E degrees in Electrical Engineering from the University of Michigan in 2006 and 2008. His research interests are in ensuring the correctness of digital hardware designs, including verification, reliable system design and post-silicon validation.

David Fick (S'08) is a Ph.D. candidate at the University of Michigan, Ann Arbor. He works with Professors Dennis Sylvester and David Blaauw in the Michigan Integrated Circuits Lab (MICL). His research interests include fault tolerance, adaptive circuits and systems, and 3D integrated circuits.

Valeria Bertacco (S'95–M'03) received the Laurea degree in computer engineering from the University of Padova, Italy, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 2003. She is an associate professor of electrical engineering and computer science at the University of Michigan. She joined the faculty at Michigan after being at Synopsys for four years. Her research interests are in the areas of formal and semiformal design verification with emphasis on full design validation and digital system reliability. She is an associate editor of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems and has served on the program committees for DAC and ICCAD.

Dennis Sylvester (S'95–M'00–SM'04–F'11) received a Ph.D. from the University of California, Berkeley and is Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, where he is the Director of the Michigan Integrated Circuits Laboratory (MICL). He has published over 300 articles along with one book and several book chapters. His research interests include the design of millimeter-scale computing systems and energy efficient near-threshold computing. He holds 7 US patents and serves as consultant and advisor to electronic design automation and semiconductor firms in these areas. He is a co-founder of Ambiq Micro, a fabless semiconductor company developing ultra-low power mixed-signal solutions for wireless devices.

David Blaauw (M'00–SM'07) received the B.S. degree in physics and computer science from Duke University, Durham, NC, in 1986, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign, Urbana, in 1988 and 1991, respectively. Until August 2001, he was with Motorola, Inc., Austin, TX, as a Manager with the High-Performance Design Technology Group. Since August 2001, he has been on the faculty of the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, where he is currently a Professor. His work has focused on very large-scale integration design with particular emphasis on ultralow power and high-performance design.

Jin Hu (S'06) is a graduate student at the University of Michigan pursuing a P.hD. and studying electronic design automation (EDA). She completed her masters at University of Michigan in 2008, and her undergraduate degrees from Northwestern in 2006. Her research interests include global routing and placement, optimizations, and logic synthesis.

Gregory Chen (S'06–M'11) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Michigan in 2006 and 2009, and 2011. He currently is a member of the High-Performance Circuits research group at Intel, Hillsboro, OR. His research interests include networks-on-chip, voltage regulation, and energy harvesting.